

Fig. 1: Memory interface

```

XPPPpreloadConfig( __XppCfg_foo );
for (int i=0; i<1000; ++i) {
    XPPPpreload( 2, &a[i*30], 30 );
    XPPPpreload( 0, &b[i*200], 200 );
    XPPPpreloadClean( 5, &c[i*10], 10 );
    XPPExecute( );
}

/* Other RISC computations ...
In the meanwhile the burst preloads and
the previous configuration are running;
The new configuration is executed as soon
as the preloads and the previous
configuration are finished.
New burst preloads can be issued
according to the FIFO length.
*/
}

Note: in all places where constants are used,
the value should actually come from a register

```

Legend:

per thread state resource
volatile (non-state) resource
write back if dirty
volatile read only resource

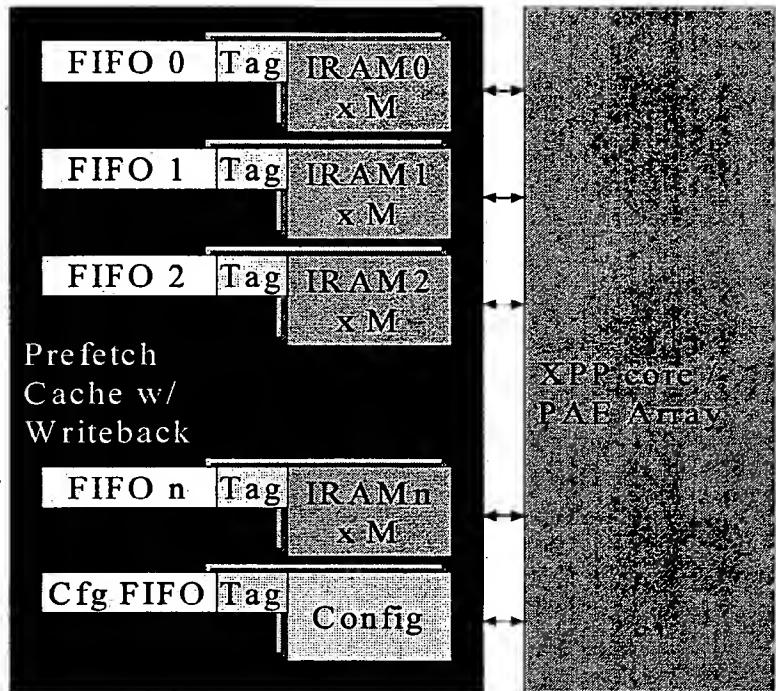


Fig. 2: IRAM & configuration cache controller data structures and usage example

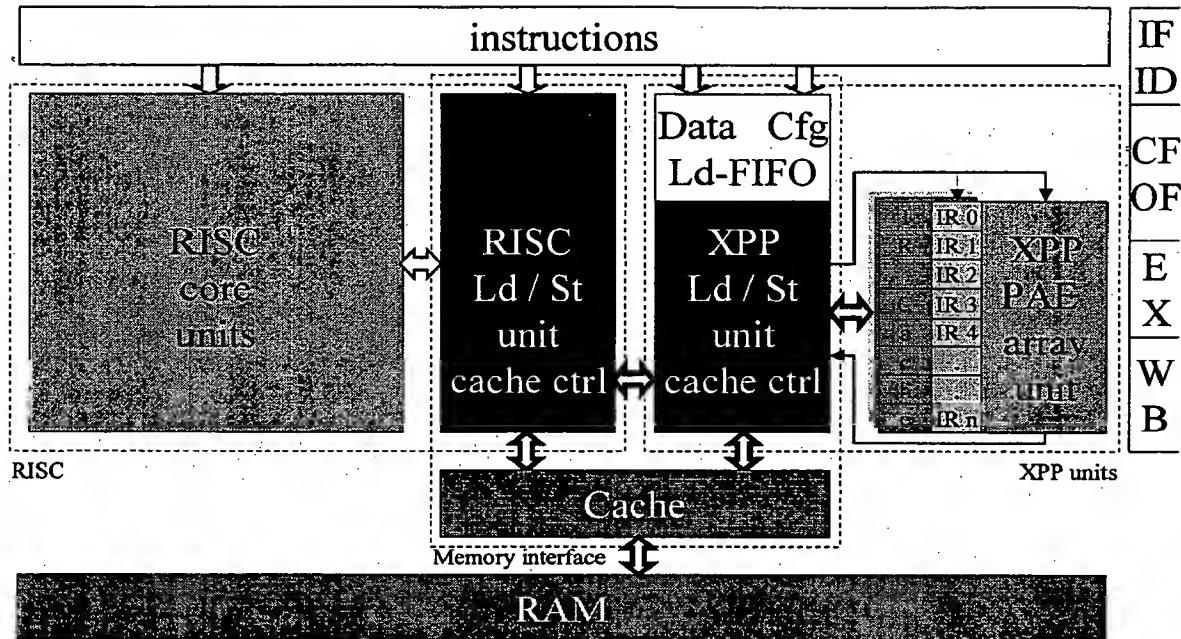


Fig. 3: Asynchronous pipeline of the XPP

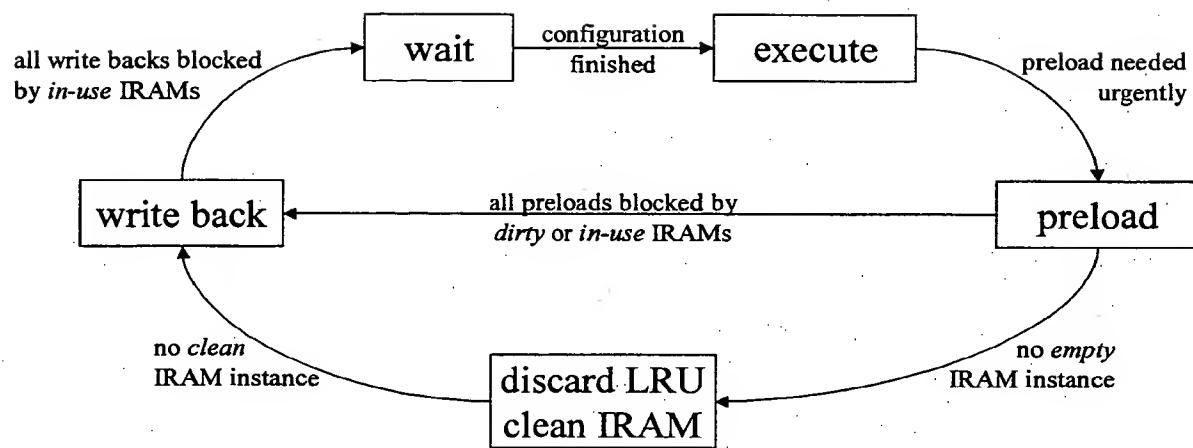


Fig. 4: State transition diagram for the XPP cache controller

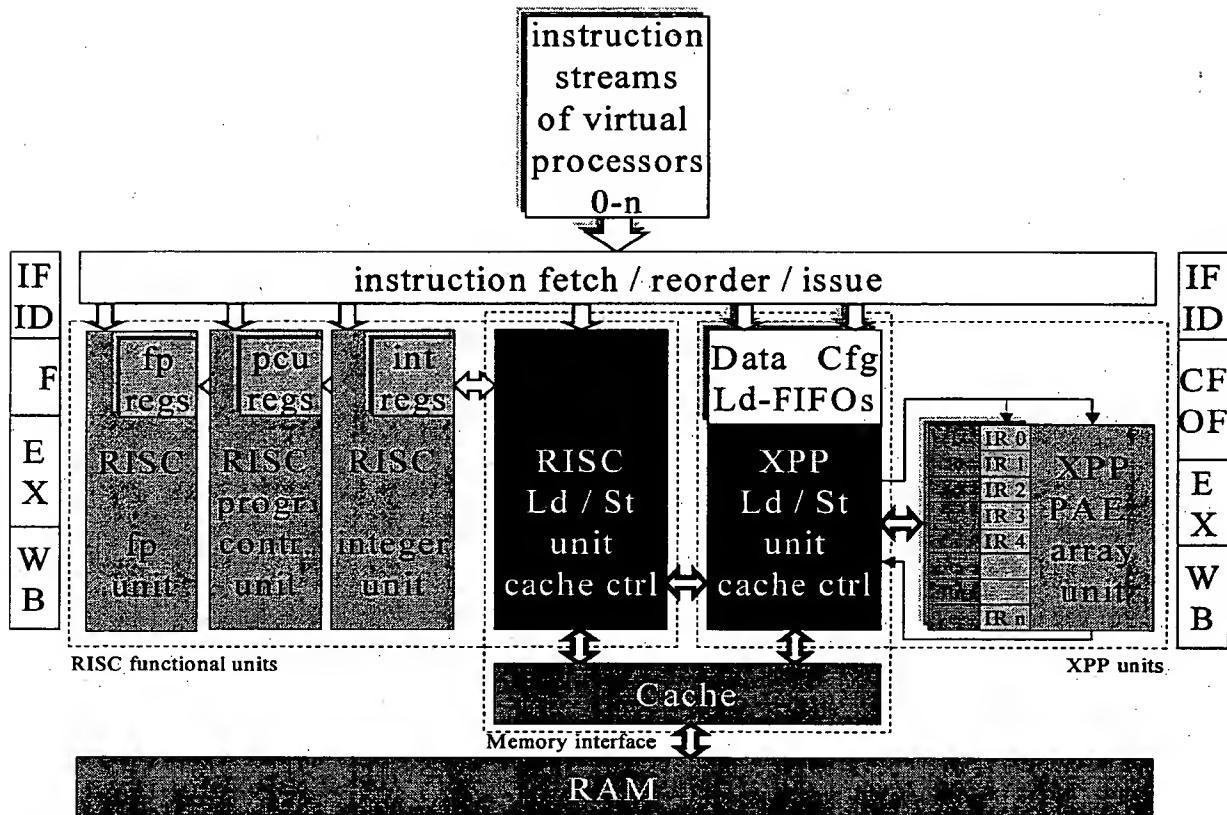


Fig. 5: Adding simultaneous multithreading

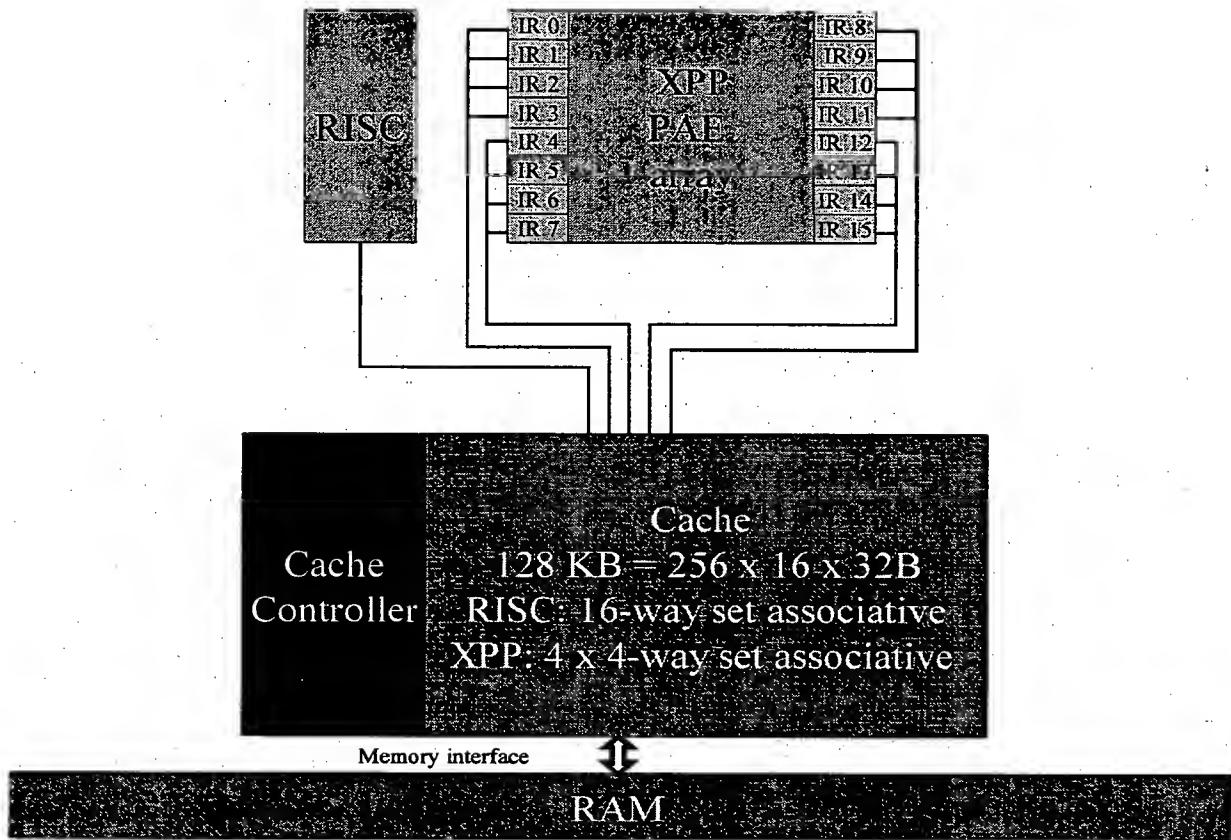


Fig. 6: Cache structure example

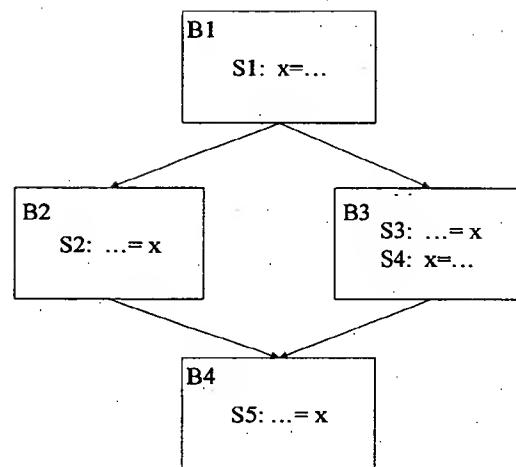


Fig. 7: Control-flow graph of a piece of program

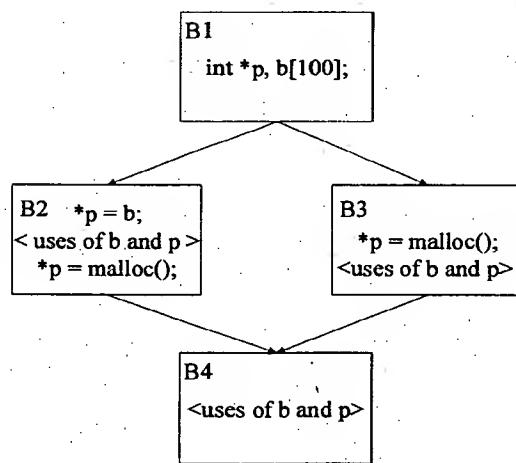


Fig. 8: Example of control-flow sensitivity

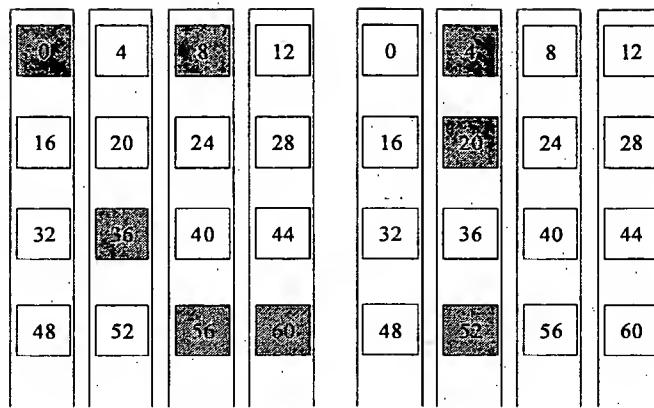


Fig. 9

```
for(j=1;j<=N-1;i++)  
    for(j=1;j<=N;j++)  
        b[i][j] = 0.25*(a[i-1][j] + a[i][j-1] +  
        a[i+1][j] + a[i][j+1]);
```



Fig. 10: Example for array merging

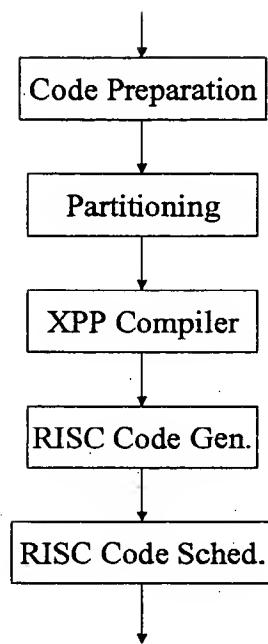


Fig. 11: Global View of the Compiling Process

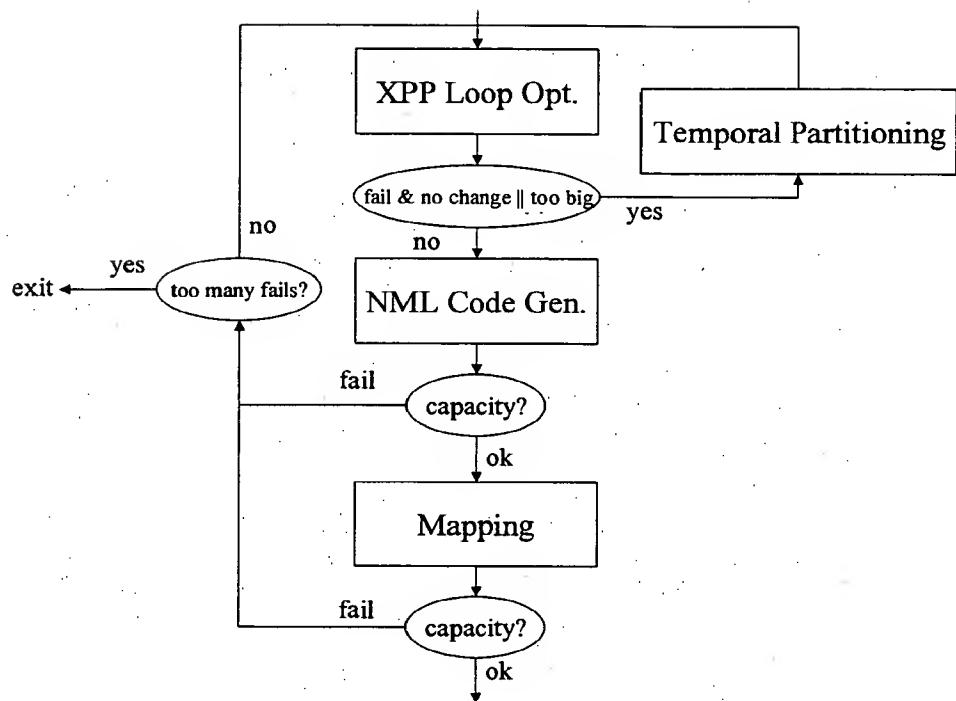


Fig. 12: Detailed Architecture of the XPP Compiler

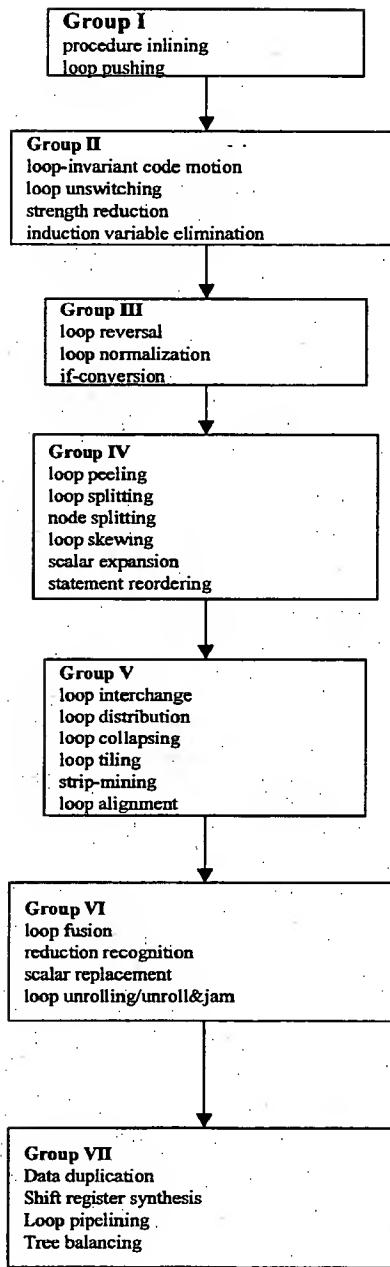


Fig. 13: Detailed View of the XPP Loop Optimization

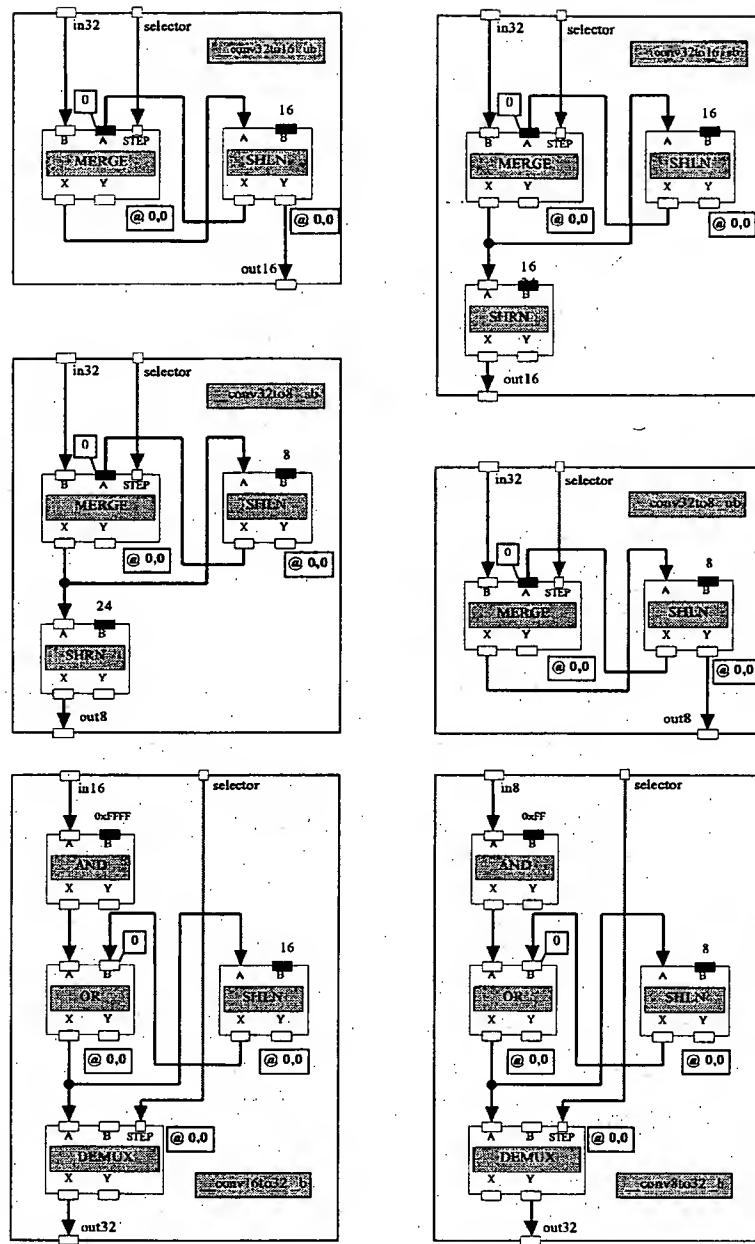


Fig. 14 :

Converter modules for conversion from and to shorter data types. The signed versions suffixed with `_sb` do correct sign extension. All modules 16-bit converters must be connected to `101010...` event streams while the `32to8`-converters must be fed with a `10001000...` sequence and the `8to32` must be fed with an a `00010001...` sequence, respectively. All modules output one packet/cycle.

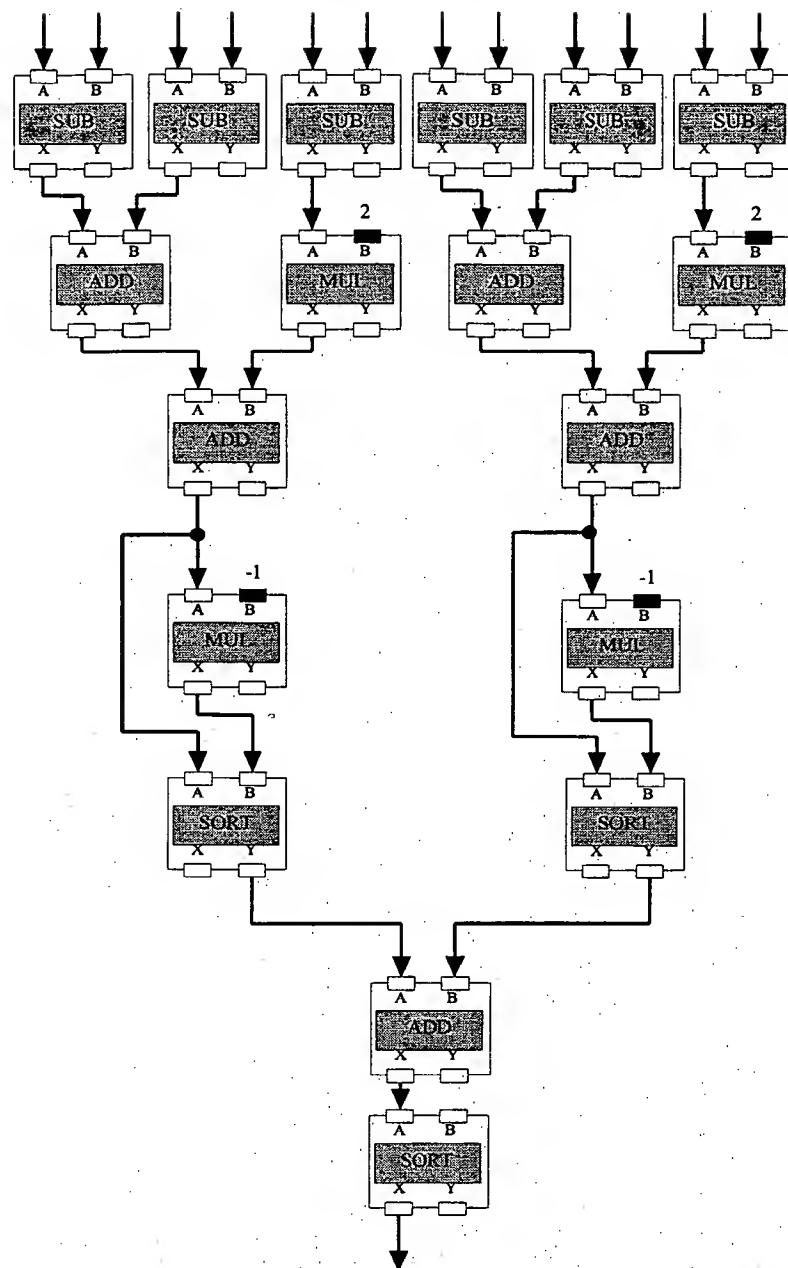


Fig. 15

The main calculation network of the edge3x3 configuration. The MULT-SORT combination does the abs() calculation while the SORT does the min() calculation.

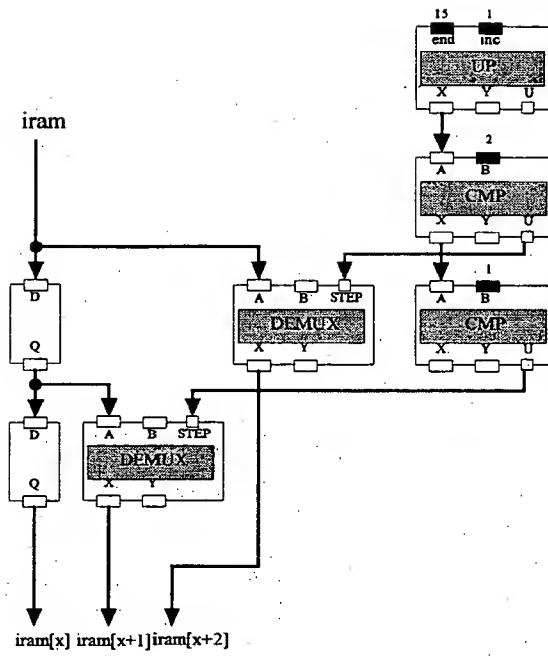


Fig. 16:

Input preparation with shift register synthesis. For each IRAM access one of these modules is generated.

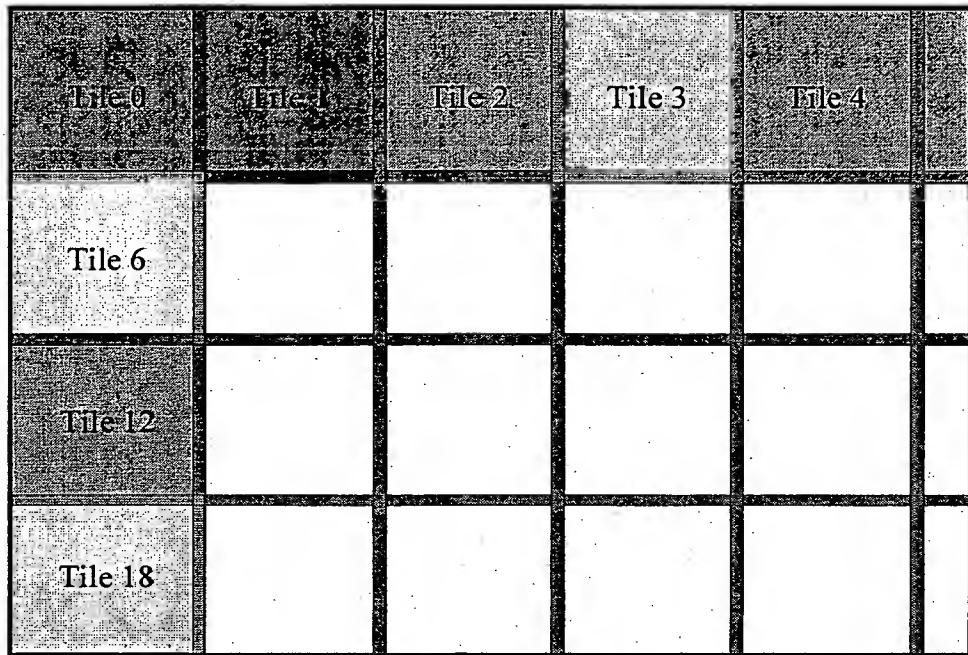


Fig. 17

A sample picture with the size 640 x 480 pixels. Without precautions loop tiling would miss the pixels on the borders between the tiles.

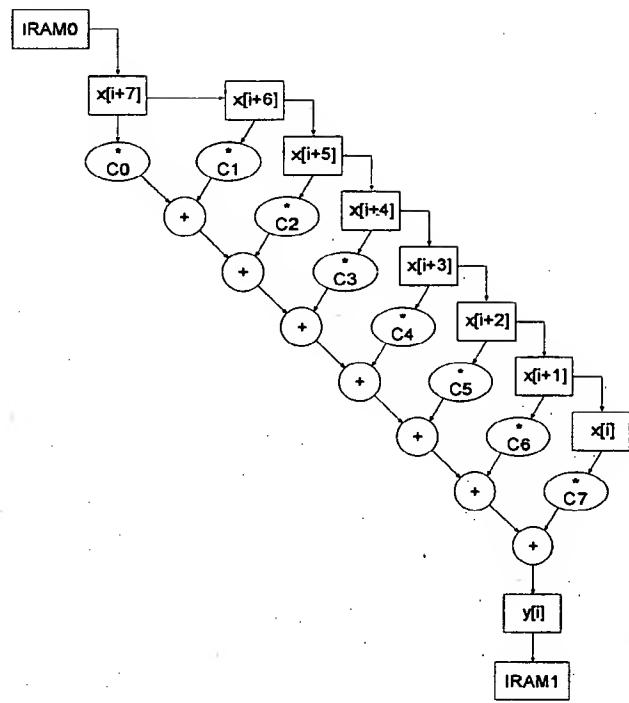


Fig. 18

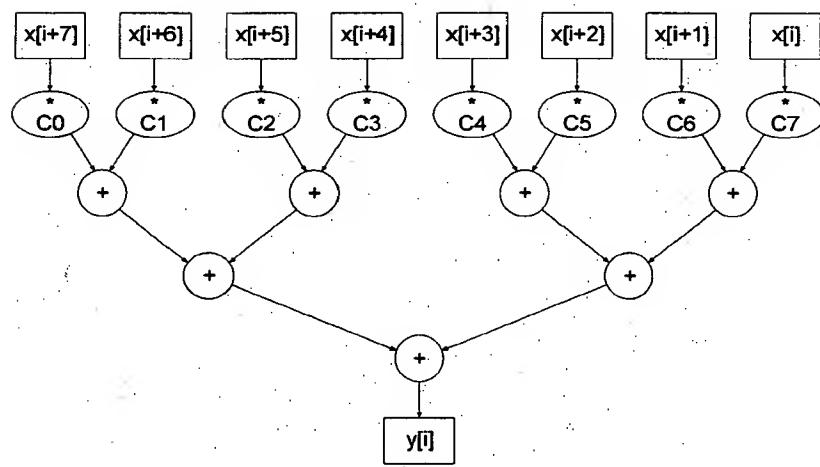


Fig. 19

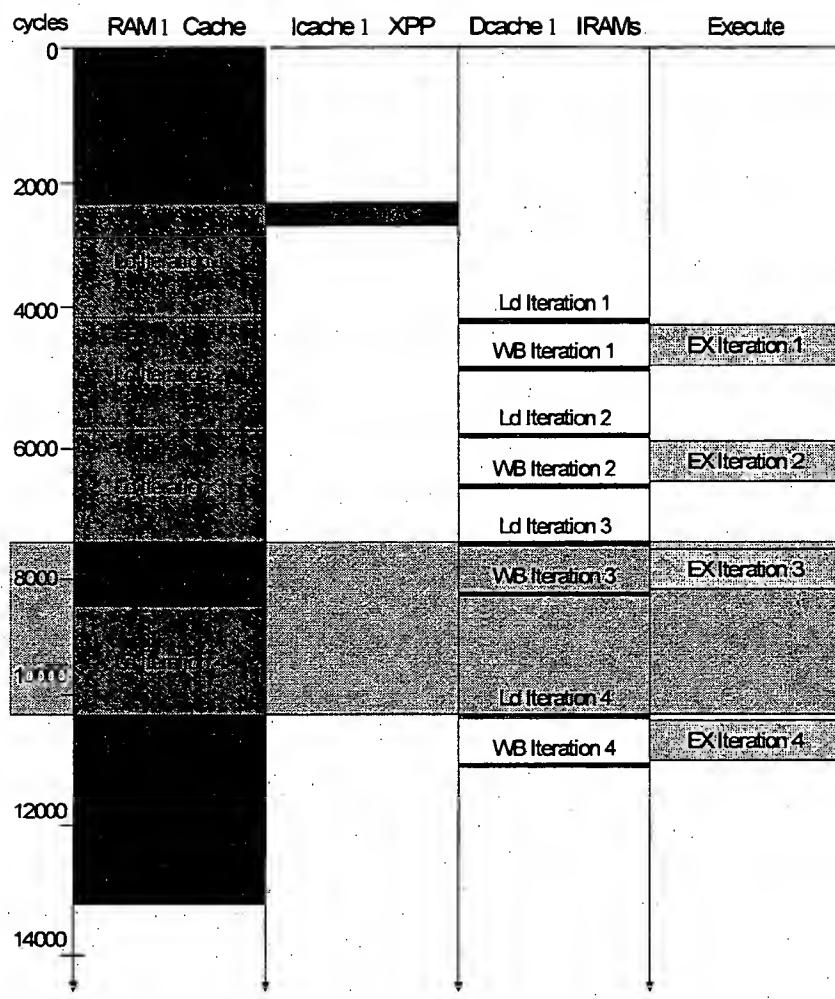


Fig. 20

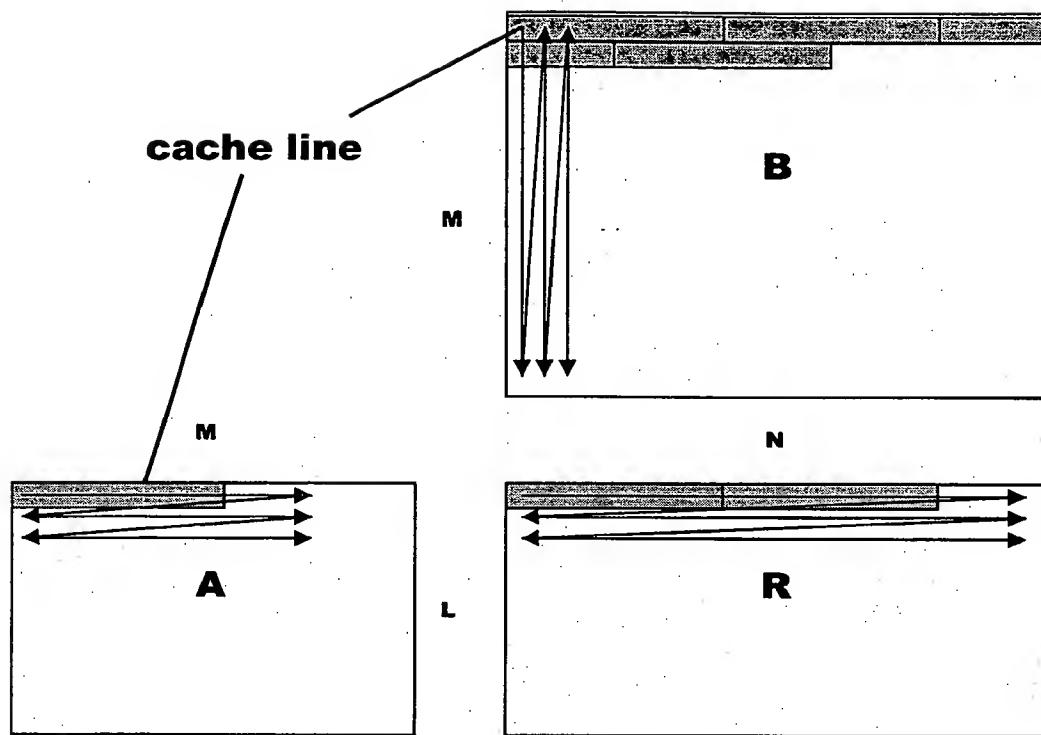


Fig. 21: The visualized array access sequences.

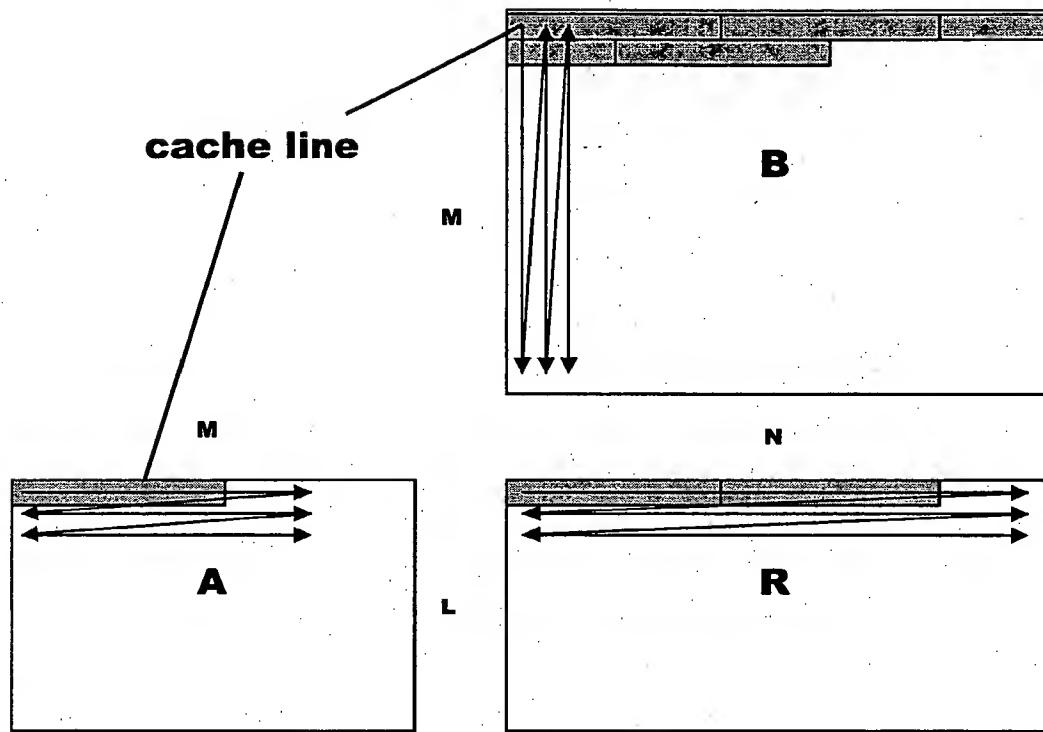


Fig. 22:

The visualized array access sequences after optimization. Here the improvement is evident, since array B is now read following the cache lines.

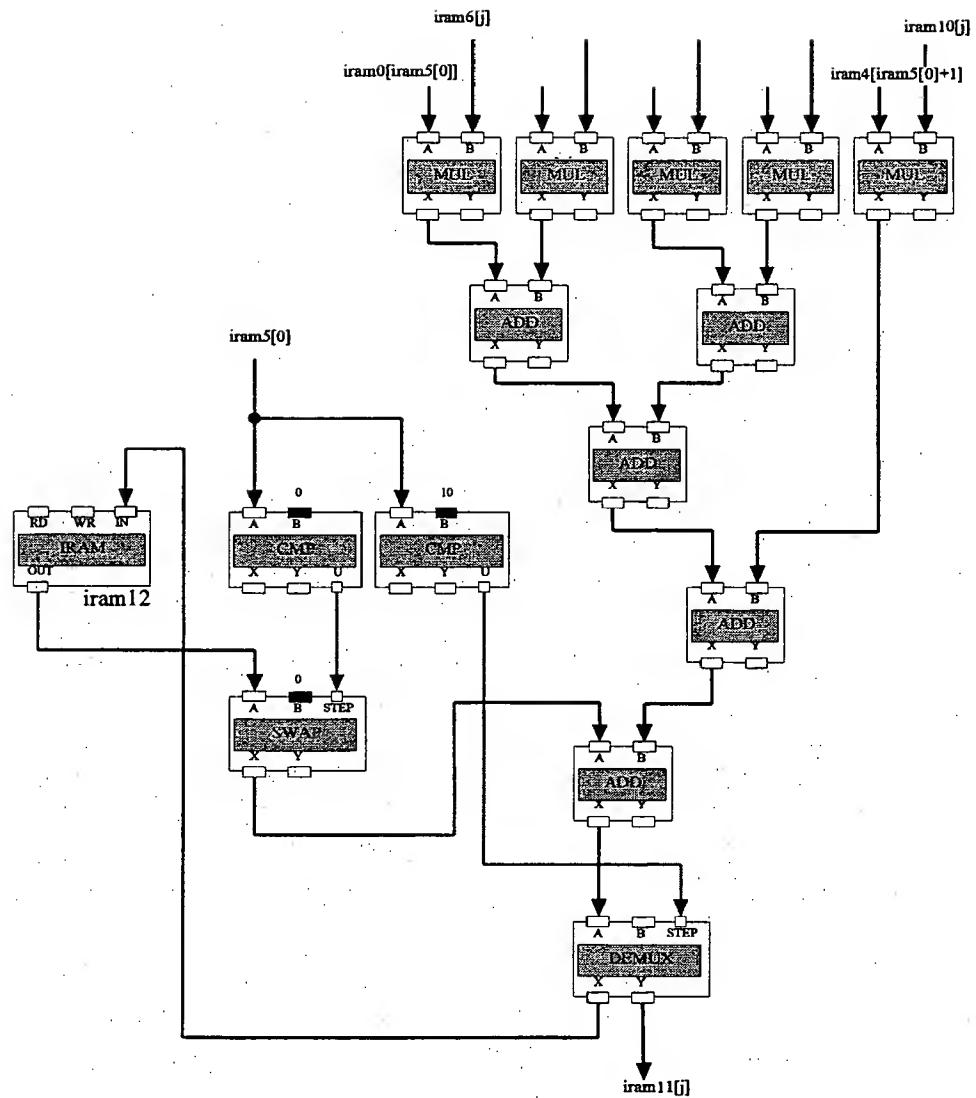


Fig. 23:

Dataflow graph of matrix multiplication after unroll-and-jam. Counters and address calculations are omitted.

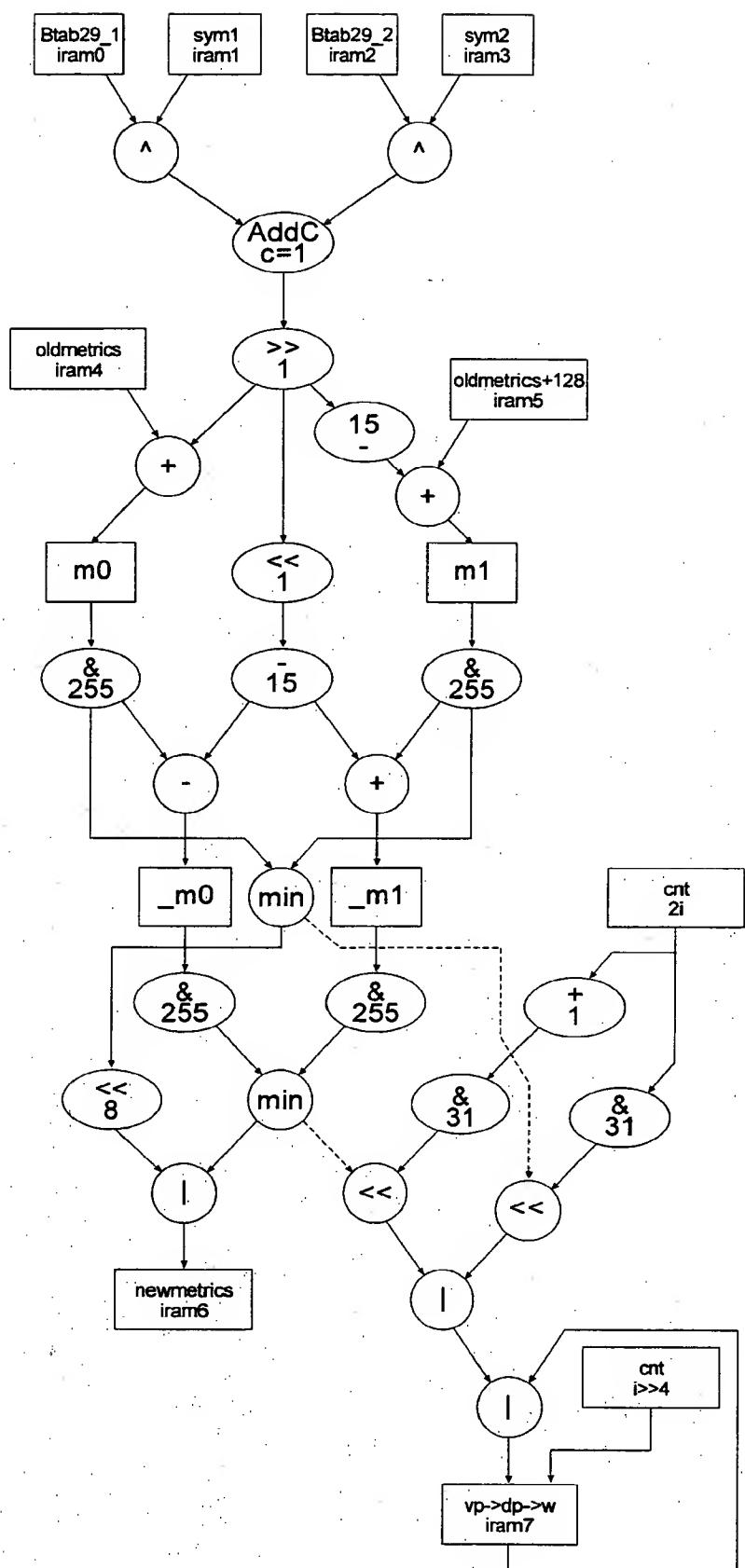


Fig. 24

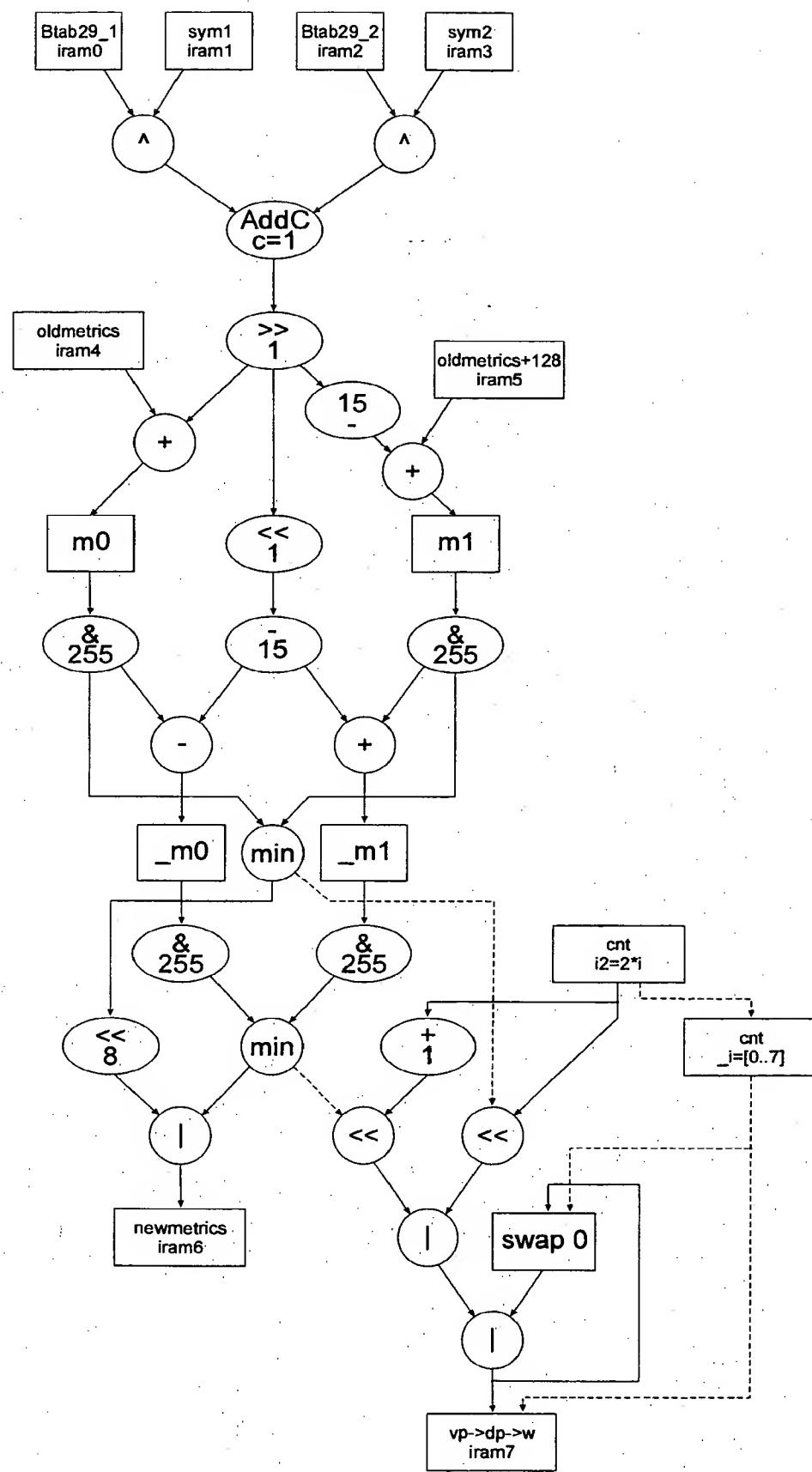


Fig. 25: The modified dataflow graph, where unrolling and splitting have been omitted for simplicity

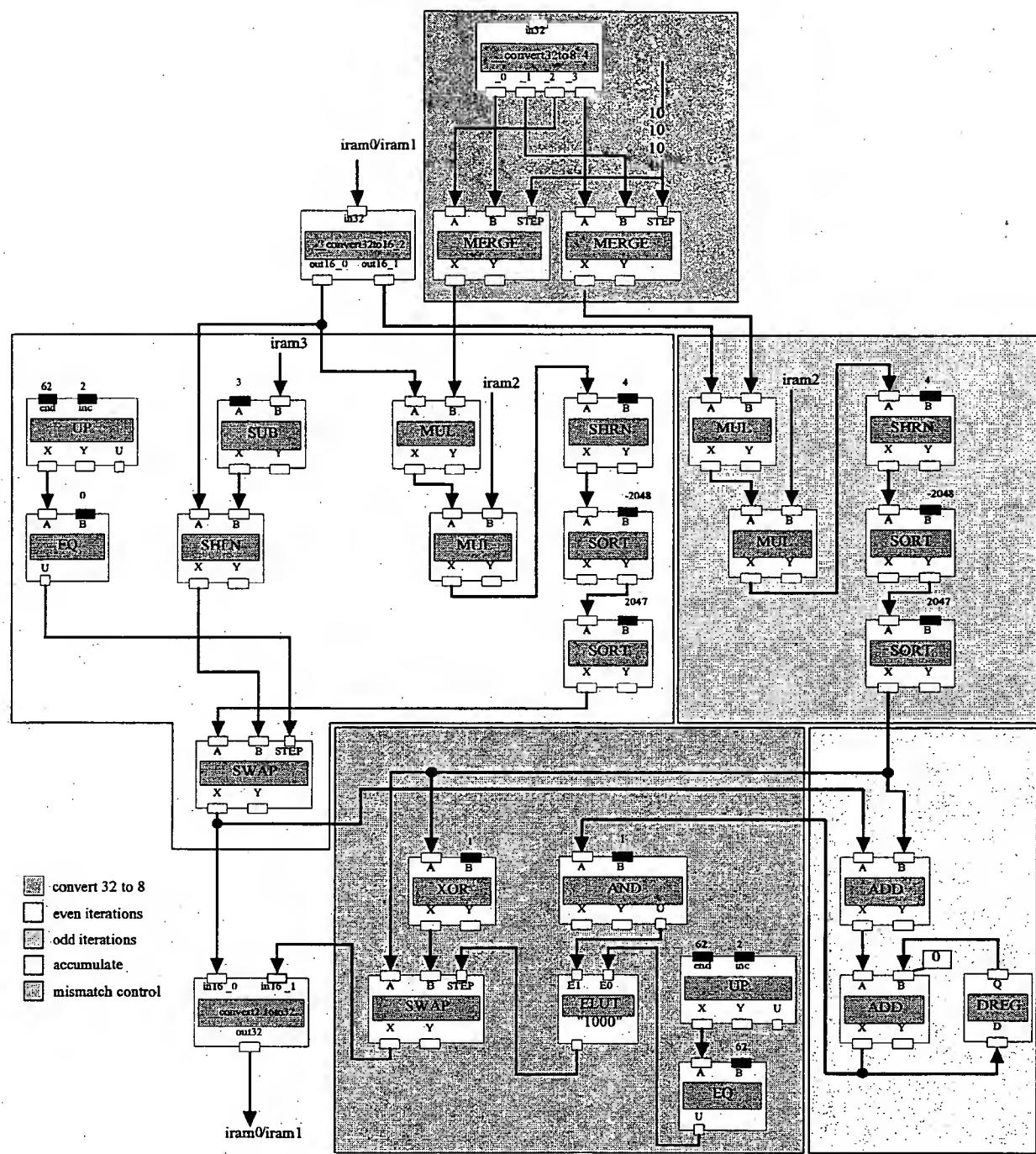


Fig. 26:

Dataflow graph of the MPEG2 inverse quantization for intra coded blocks. The yellow and green blocks were produced by partial unrolling. The difference is that the green block must no account for the special iteration value 0. The blue block does the accumulation which alters the value at iteration 64 if necessary.

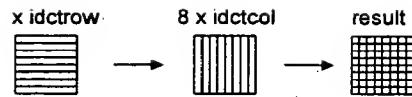


Fig. 27

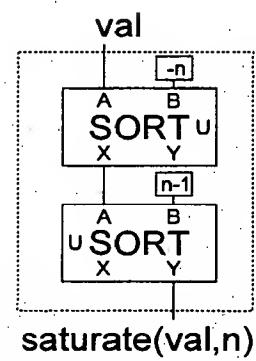


Fig. 28

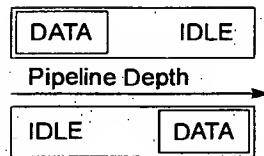


Fig. 29

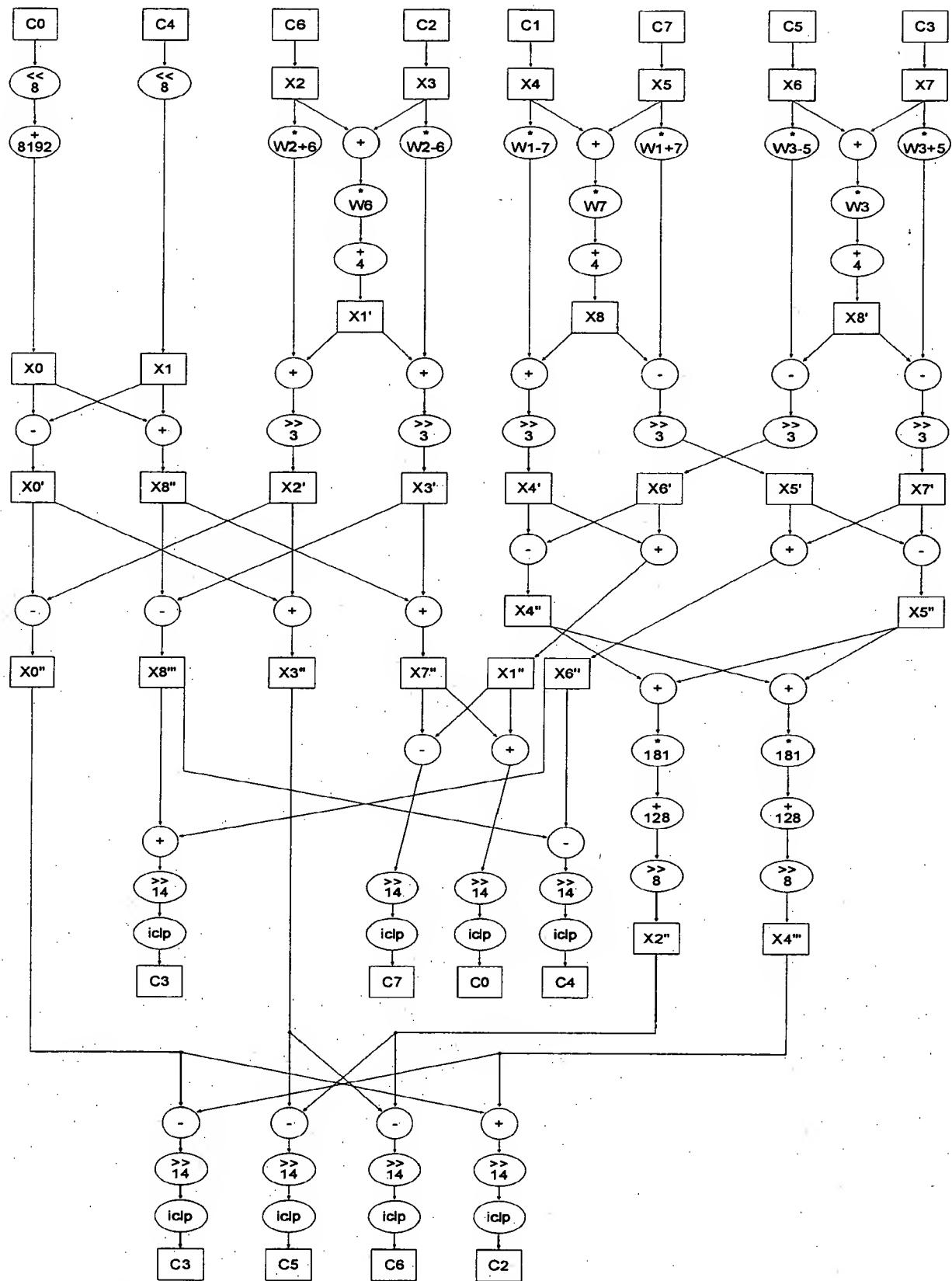


Fig. 30: Dataflow Graph of idct column processing

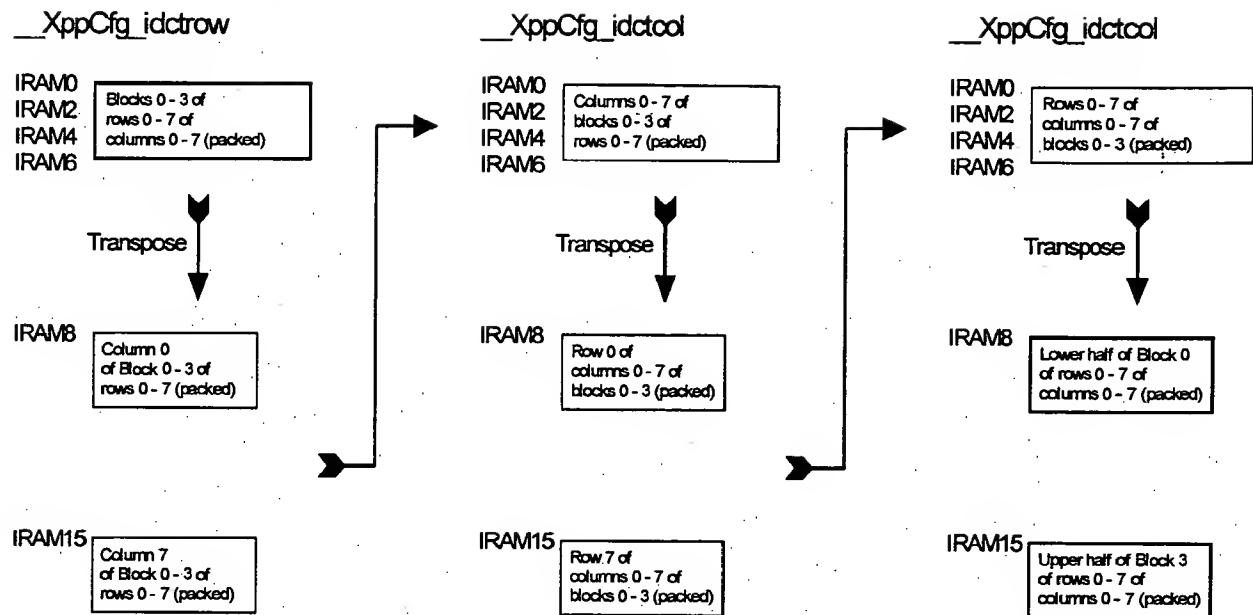


Fig. 31: Data layout transformations in idct configurations

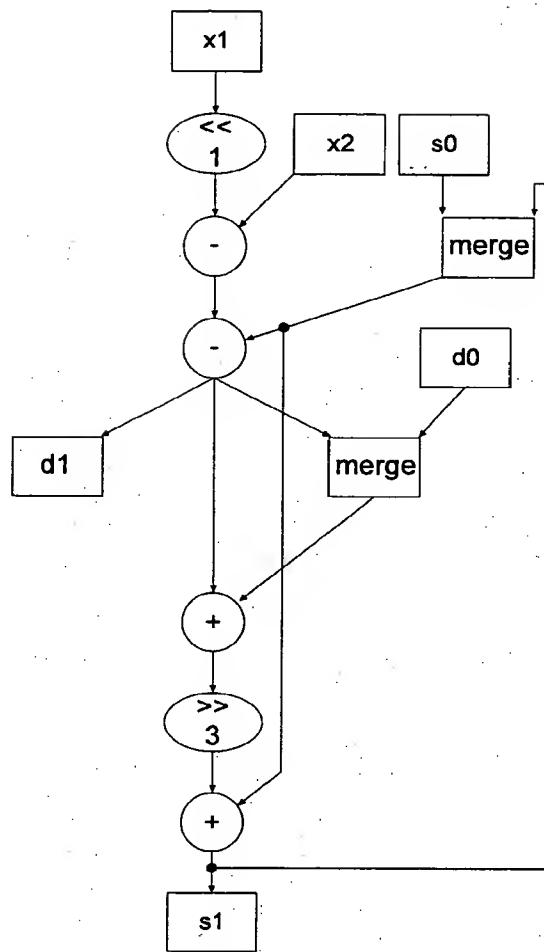


Fig. 32: Dataflow graph of the innermost loop nest.

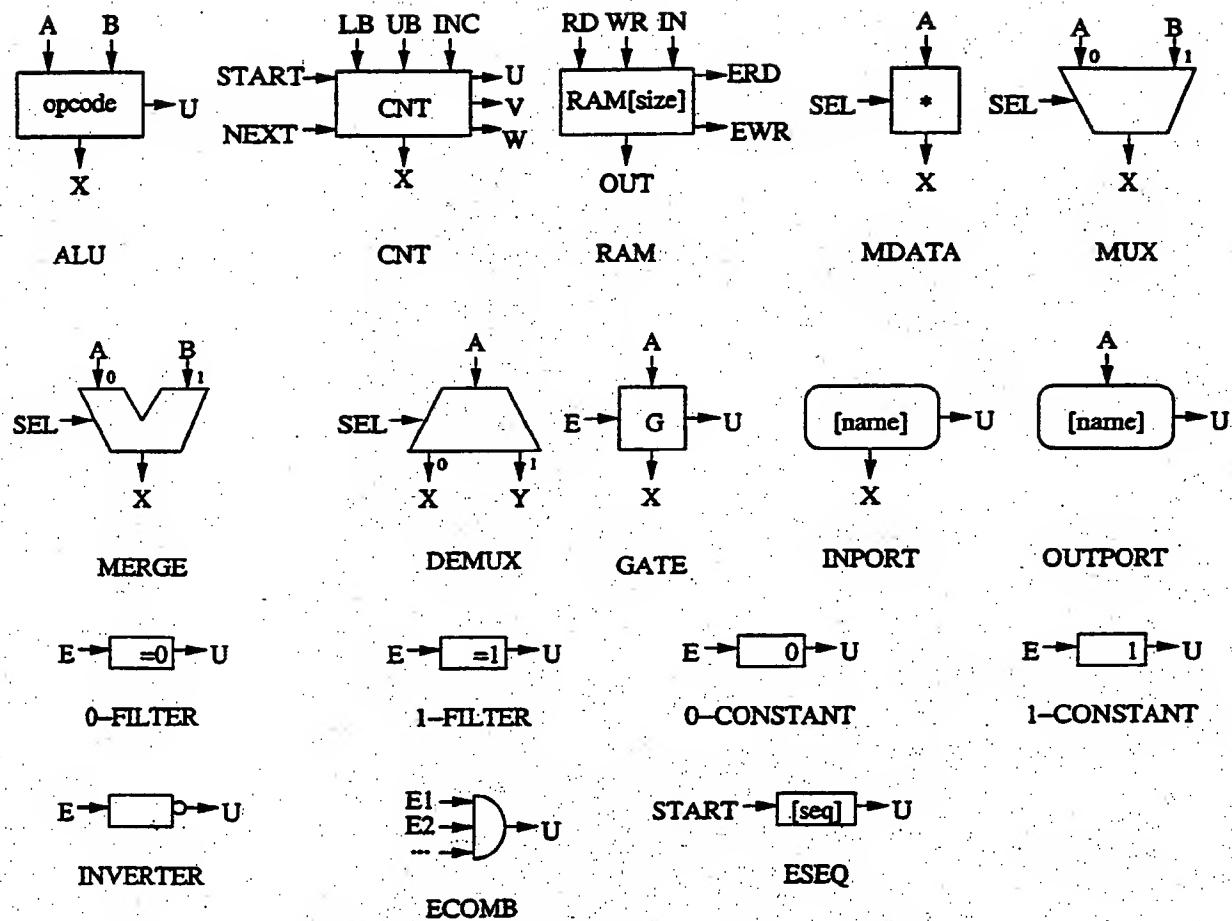


Fig. 33: Functions of an RDFP

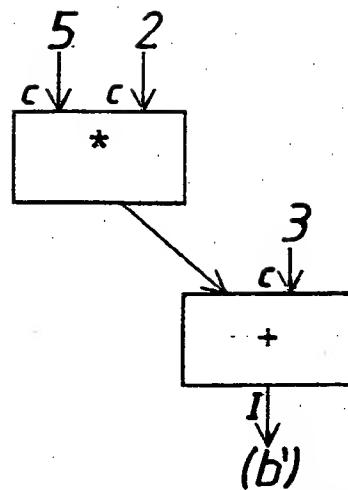


Fig. 34

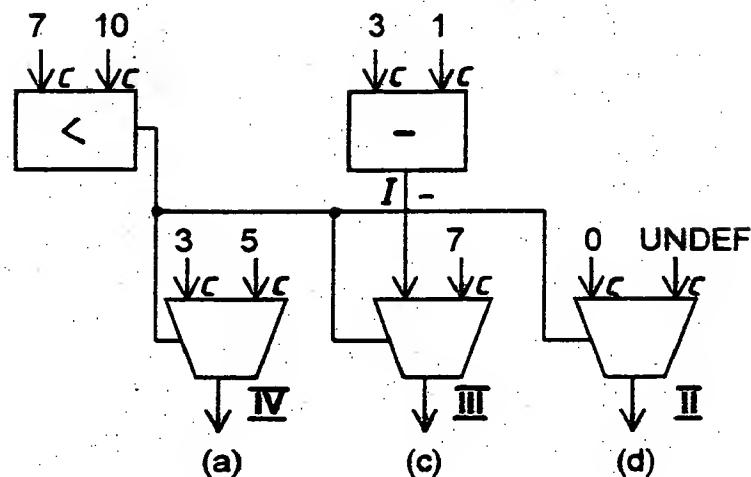


Fig. 35

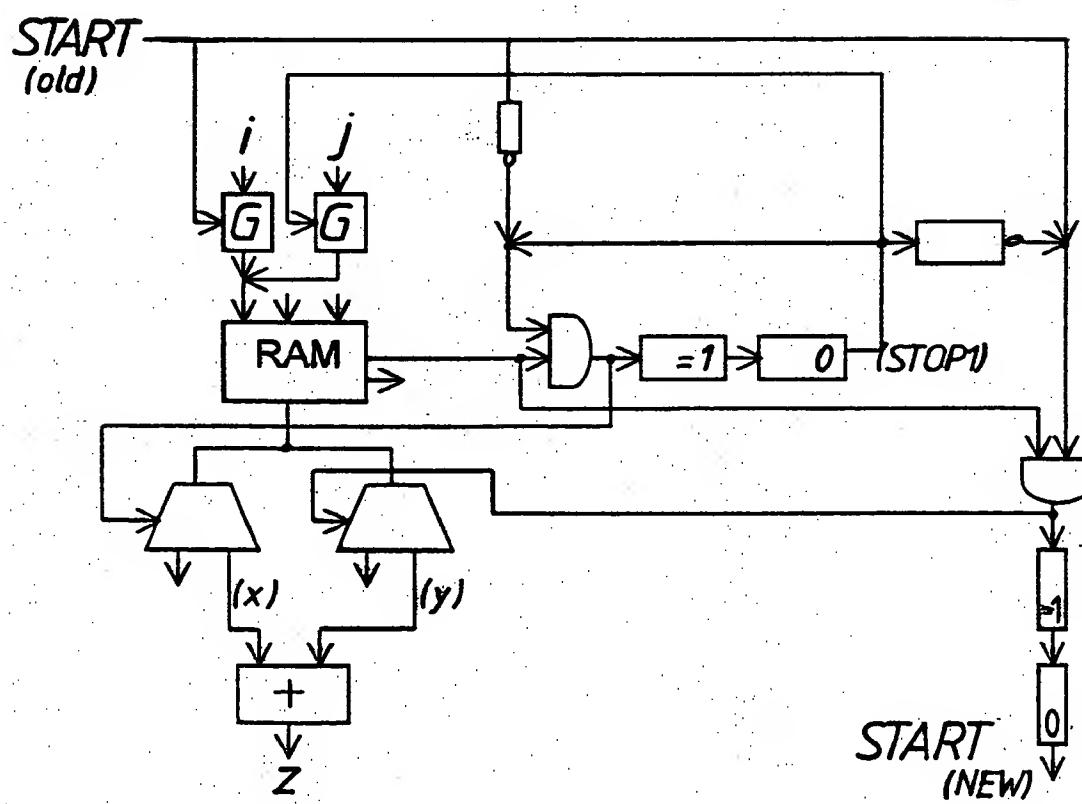


Fig. 36

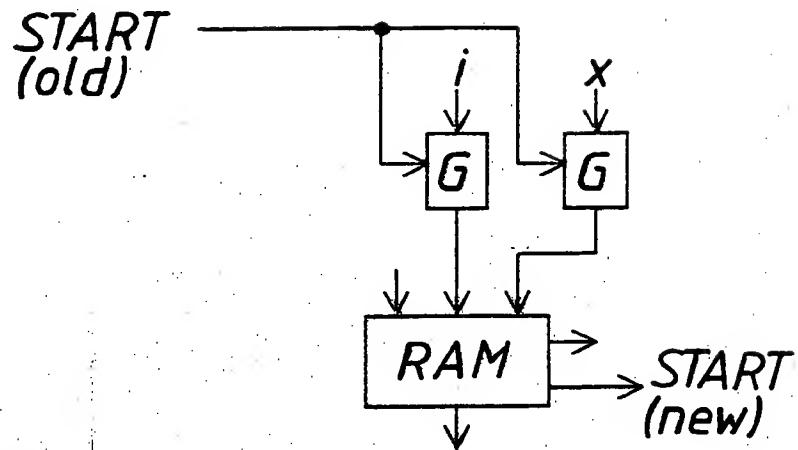


Fig. 37

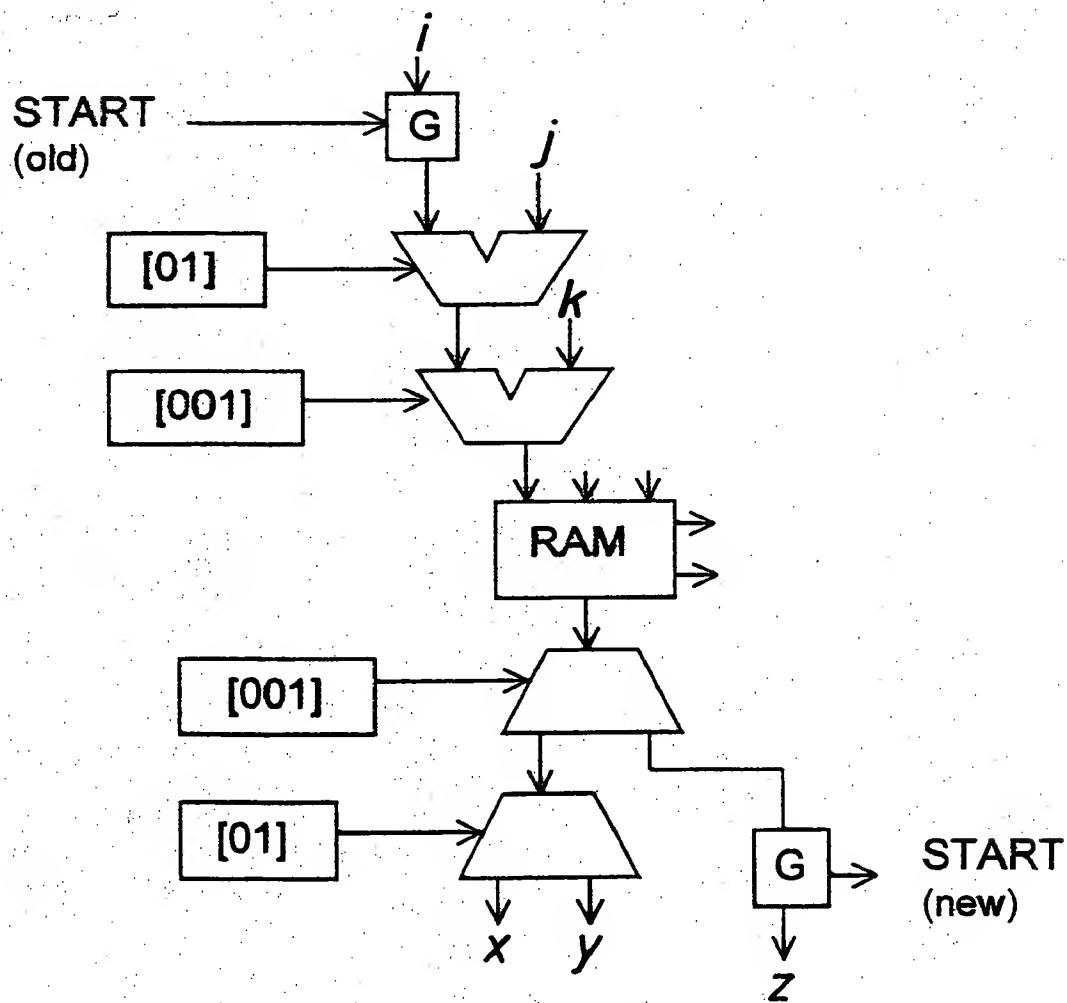


Fig. 38

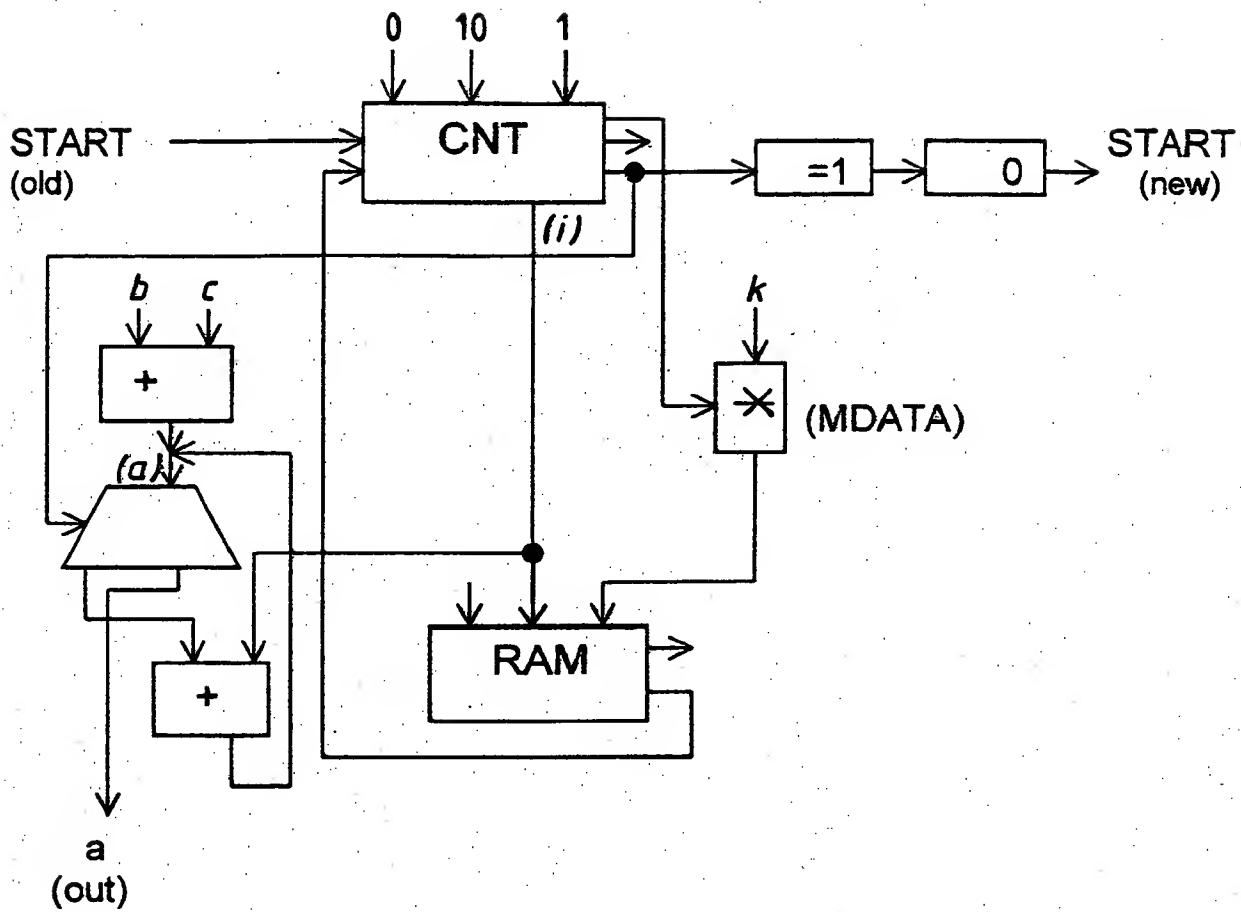


Fig. 29

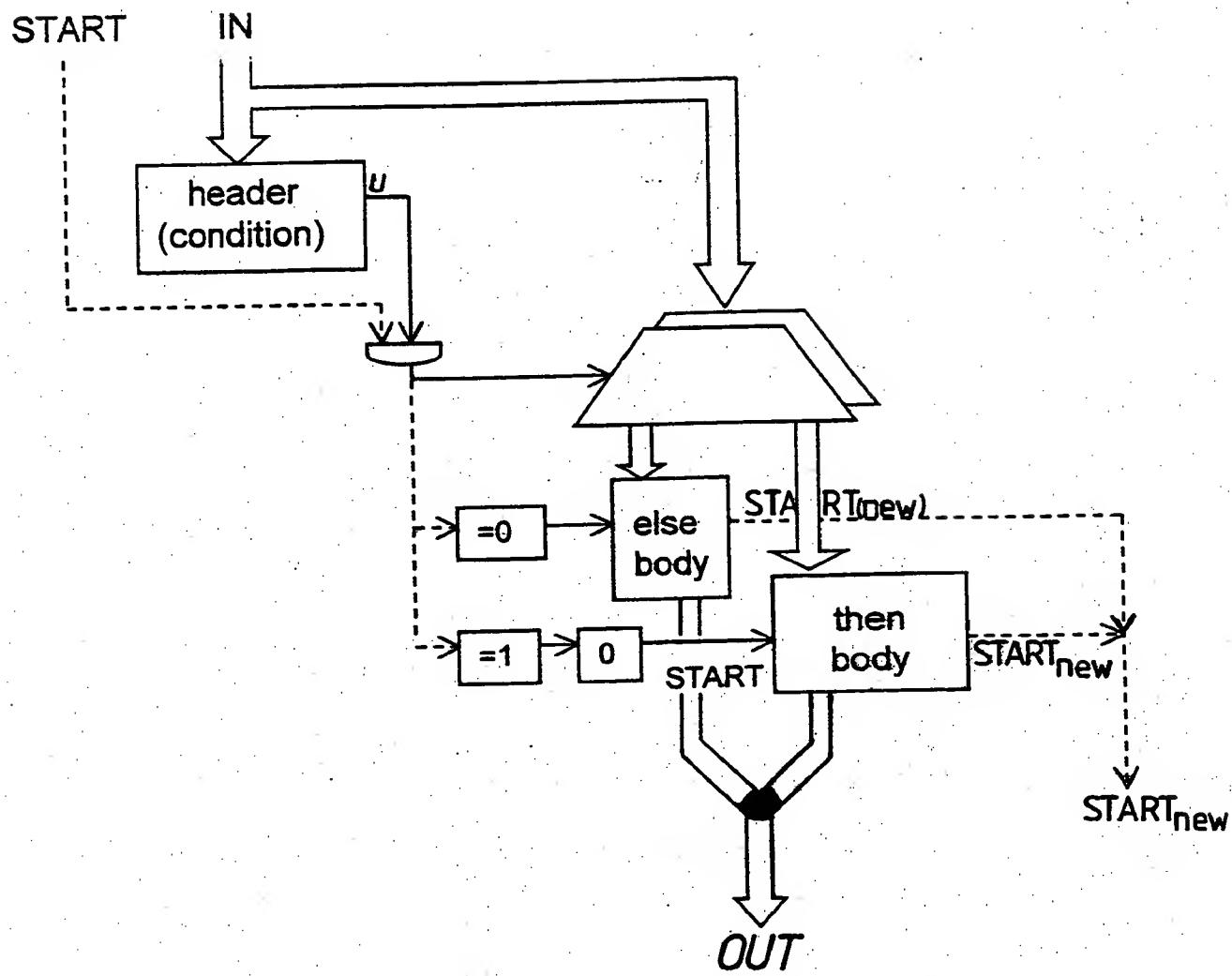


Fig.40: General Conditional Statement Template

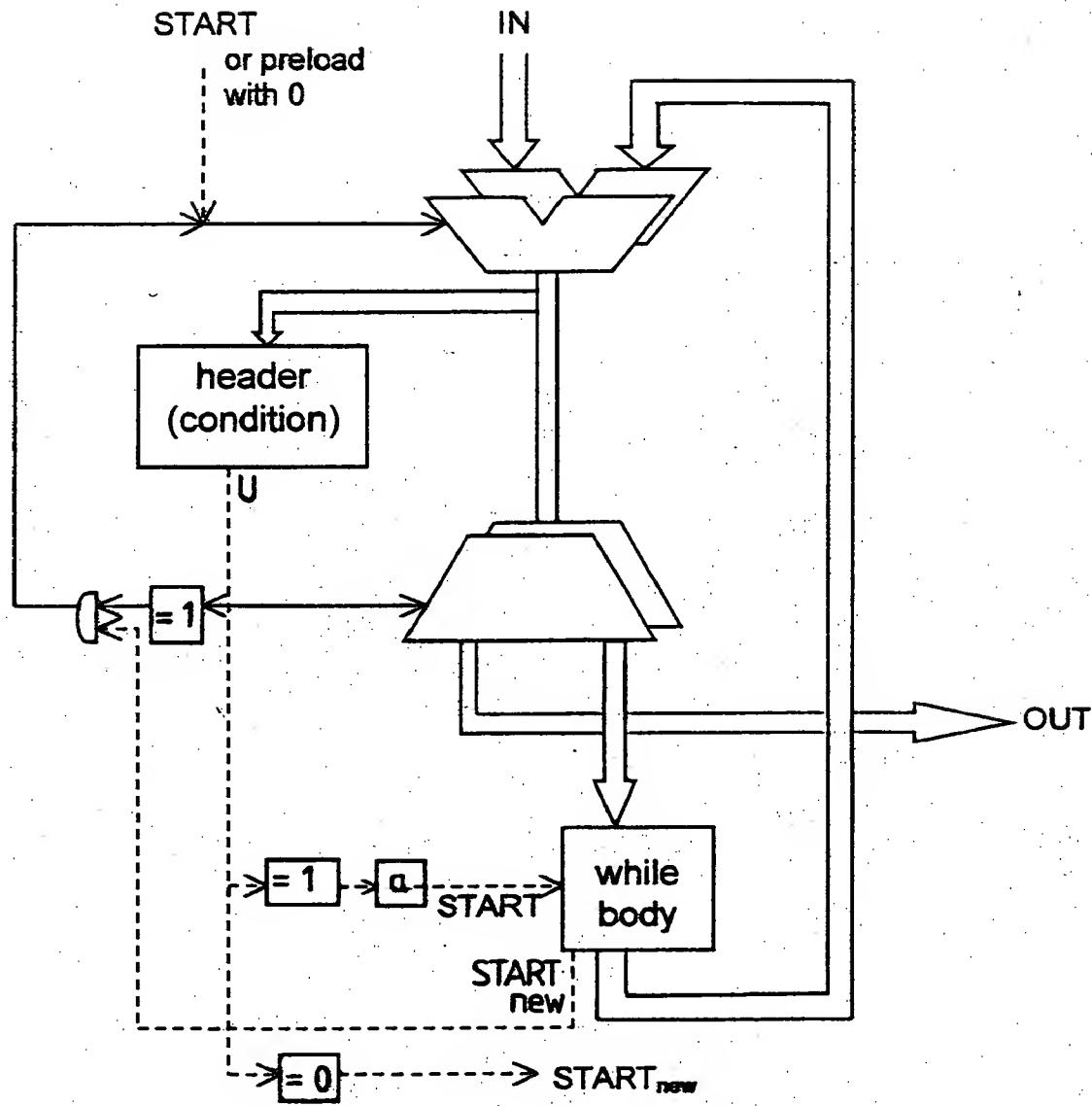


Fig. 41: While Loop Template

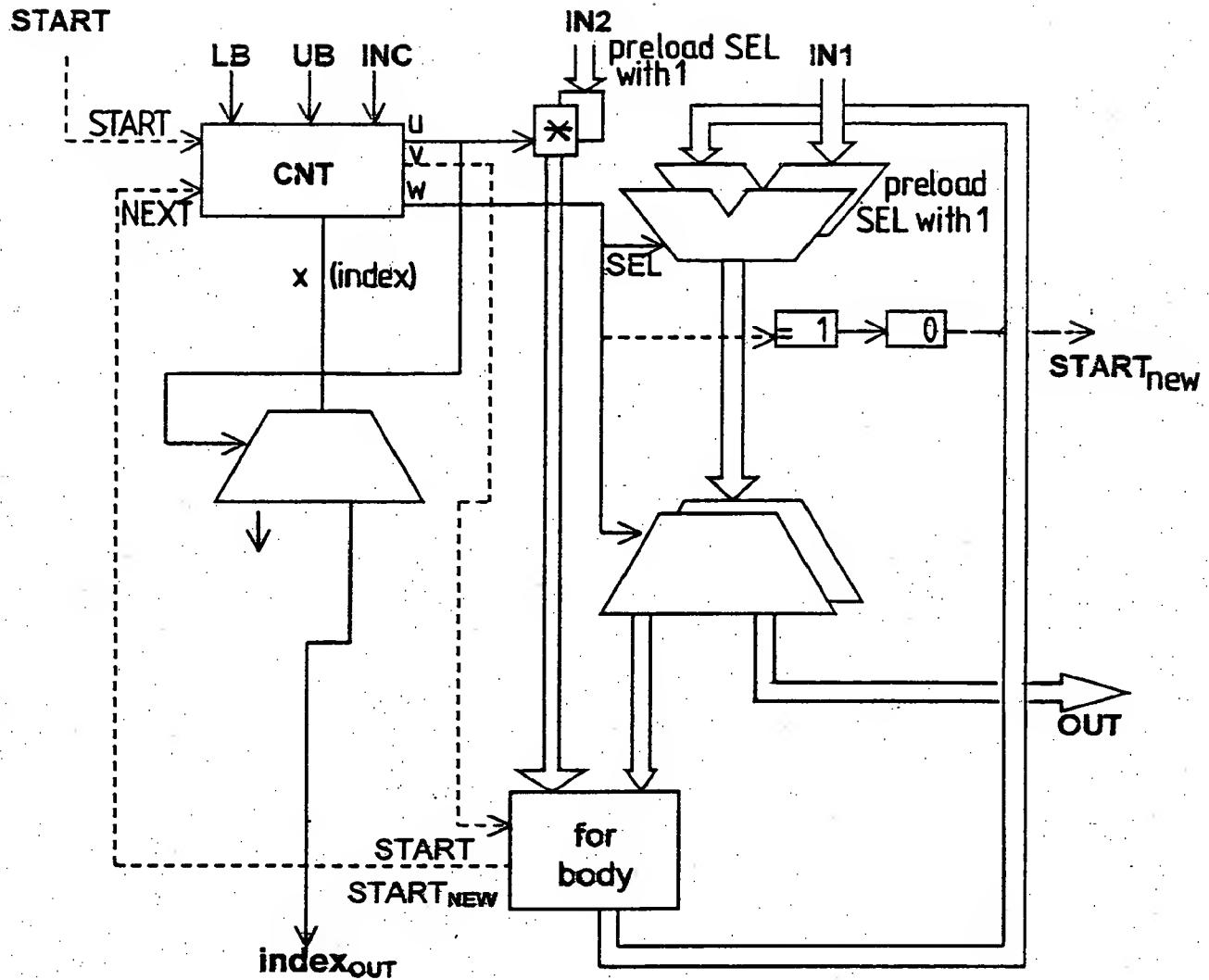


Fig. 42: For Loop Template

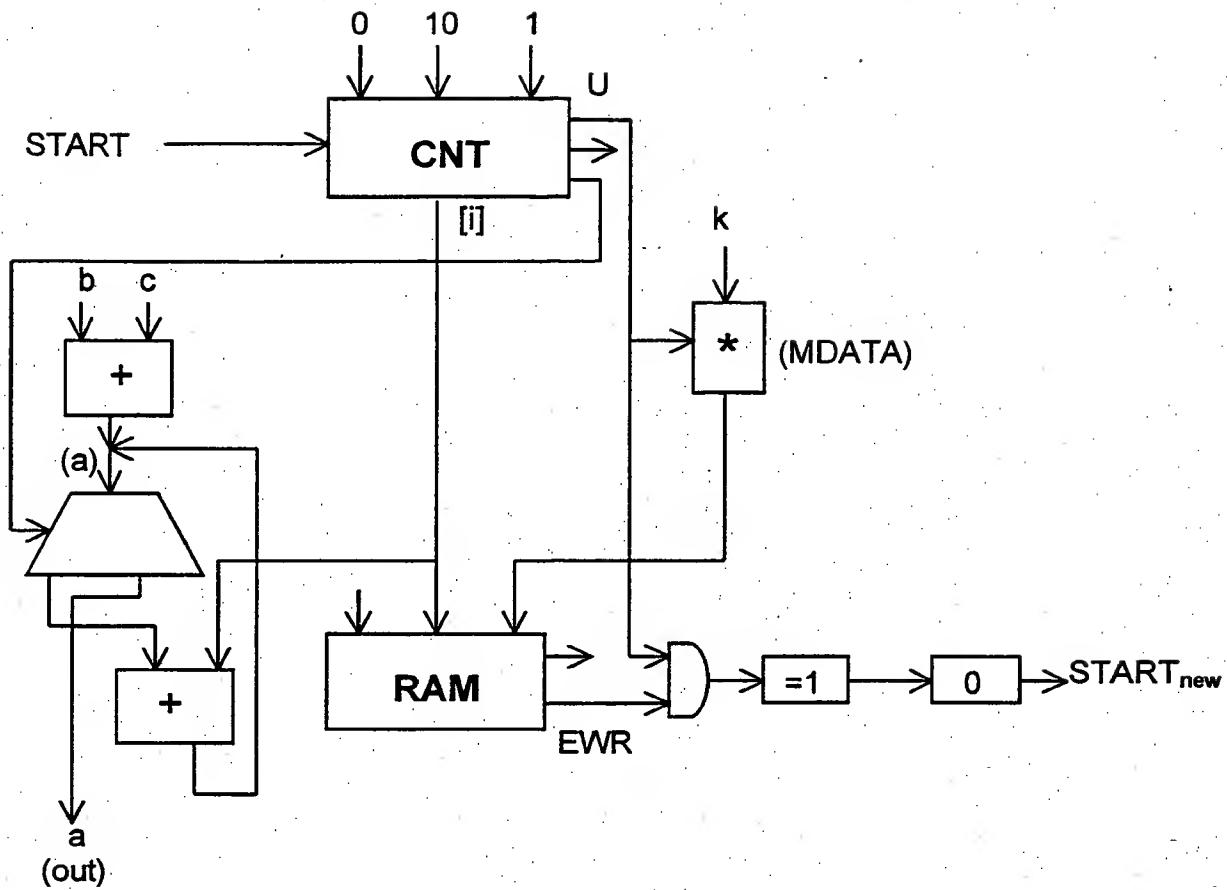


Fig. 43

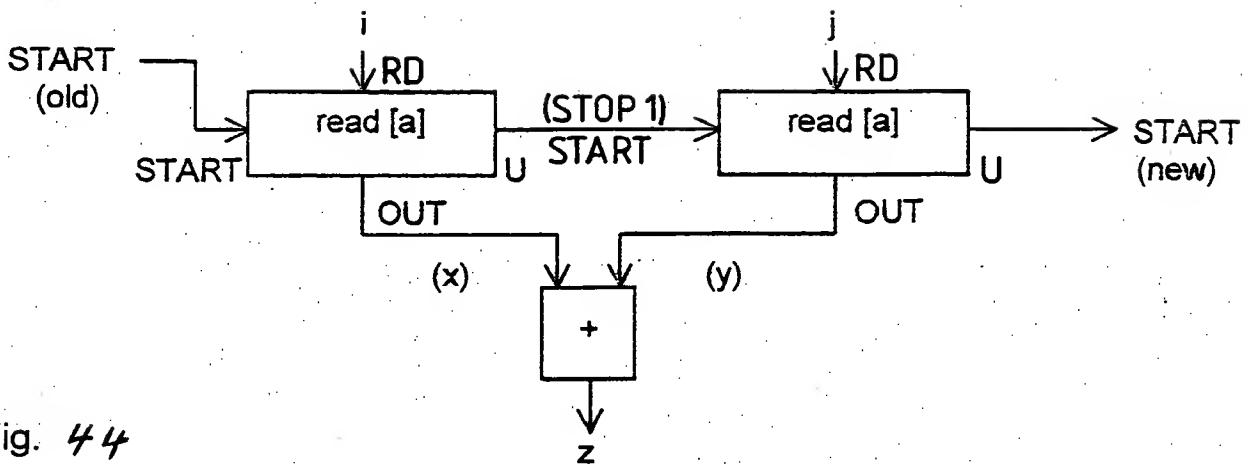


Fig. 44

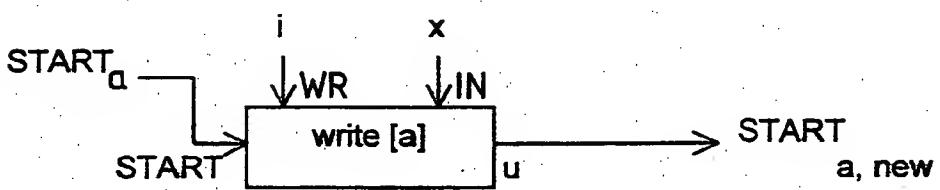


Fig. 45

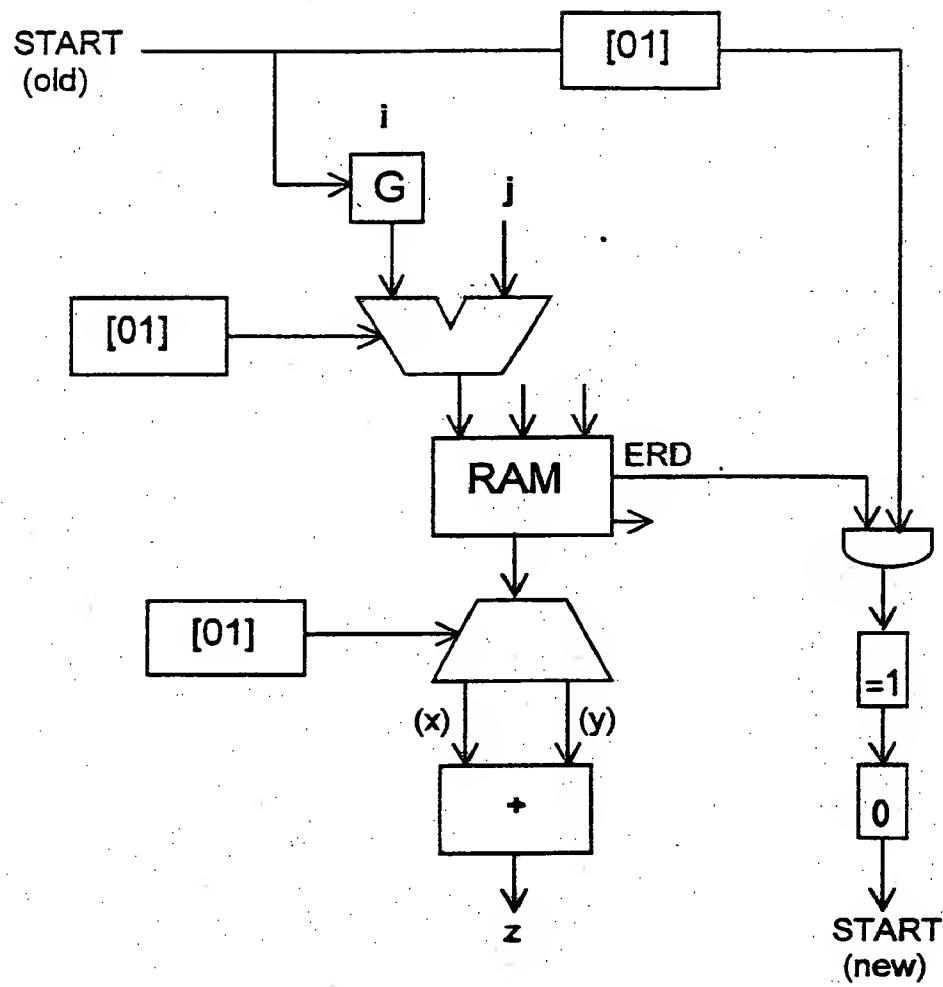


Fig. 46

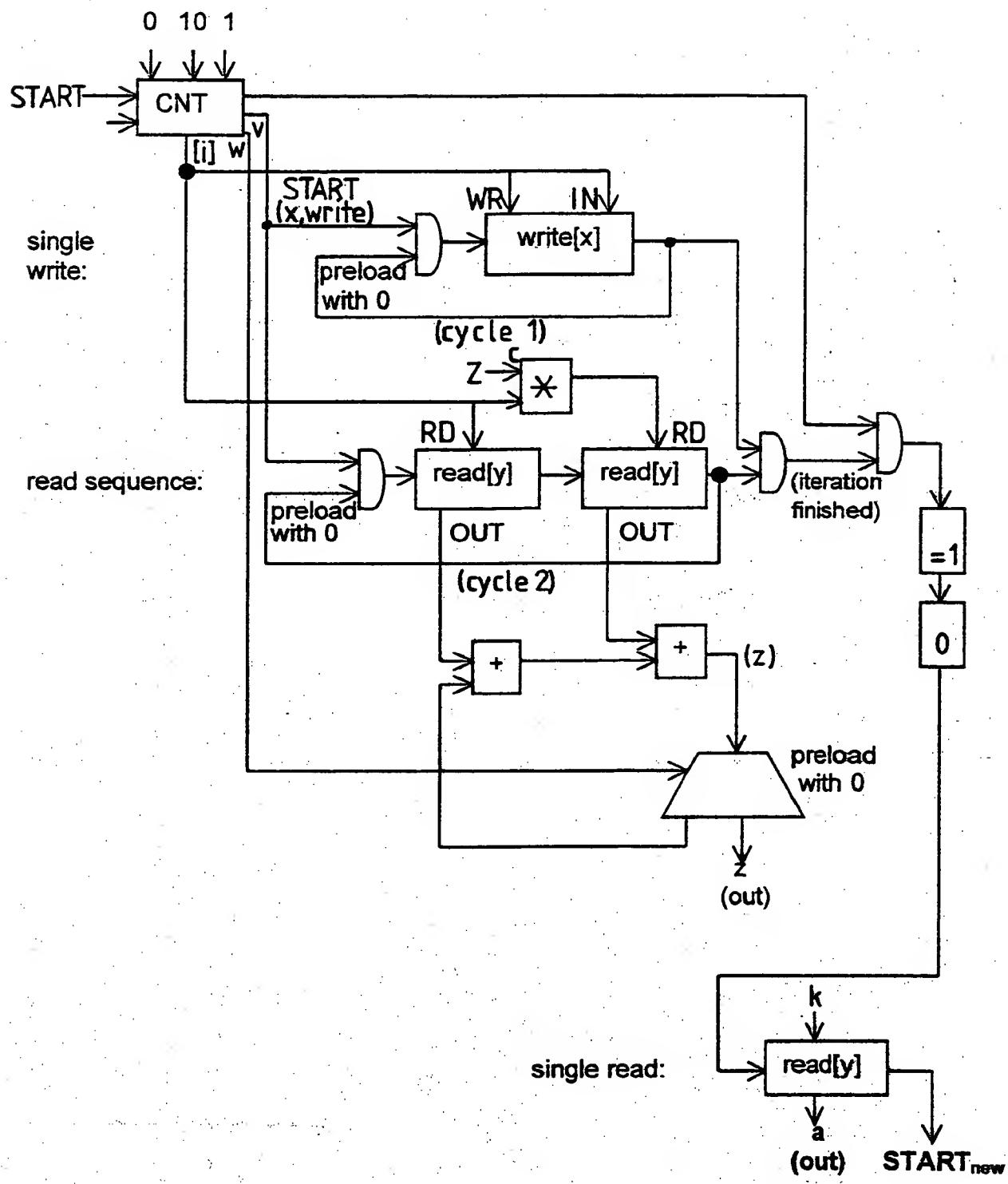


Fig. 47

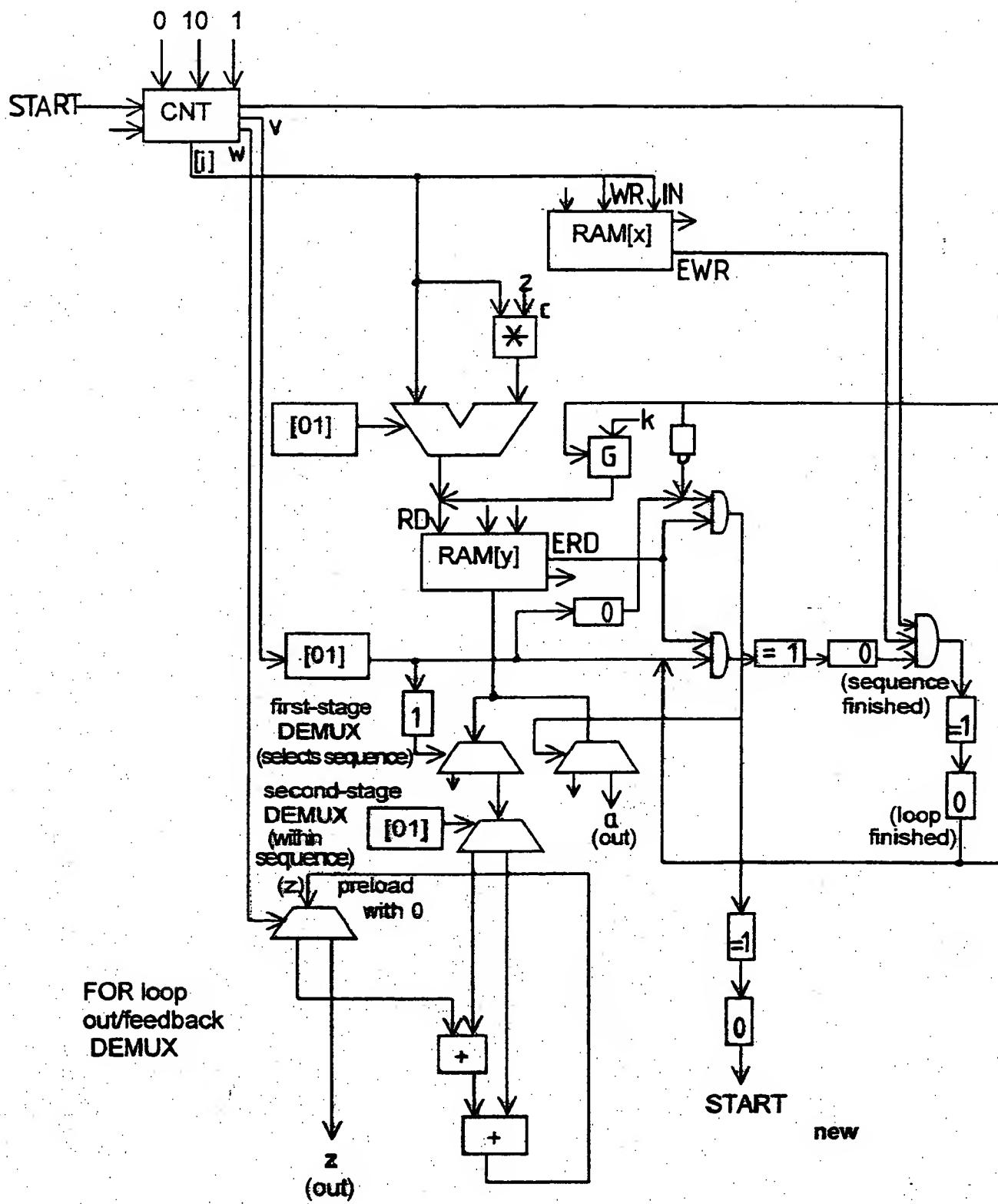


Fig. 48

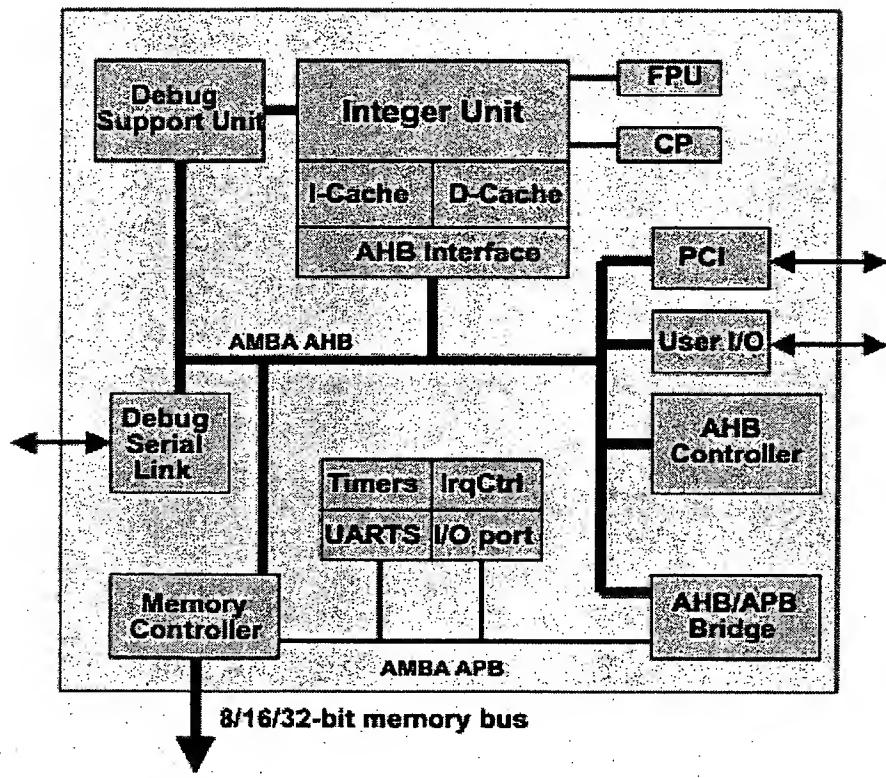


Fig. 49: LEON Architecture Overview

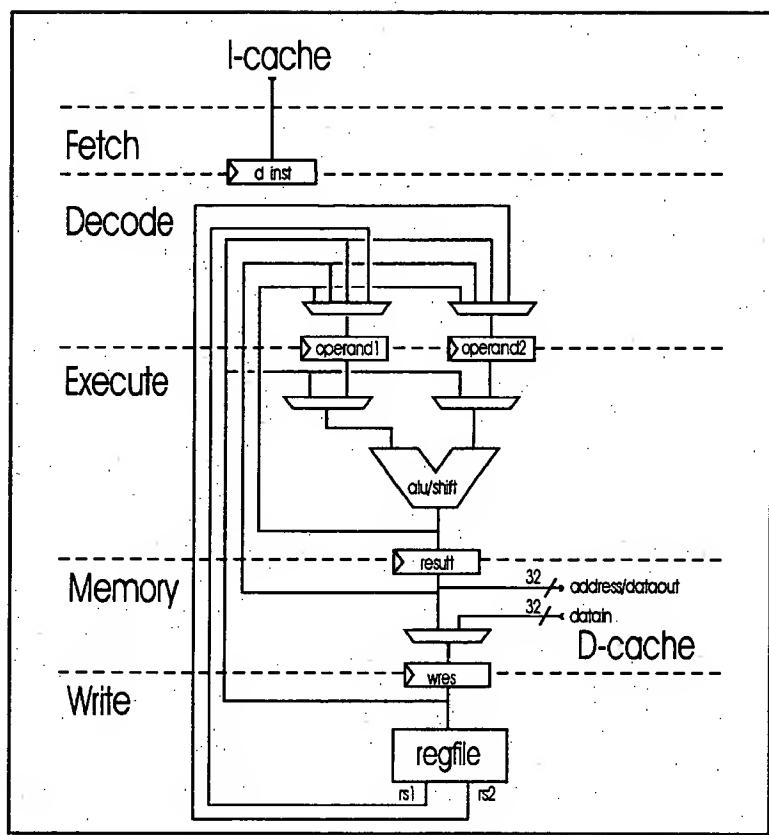


Fig. 50: LEON Pipelined Datapath Structure

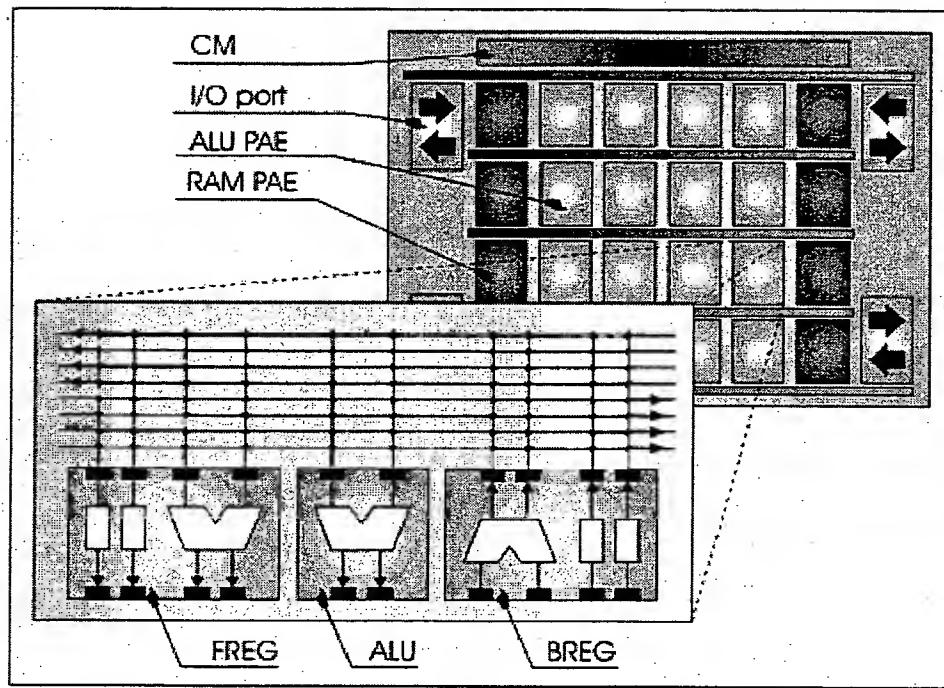


Fig. 51: Structure of an XPP device

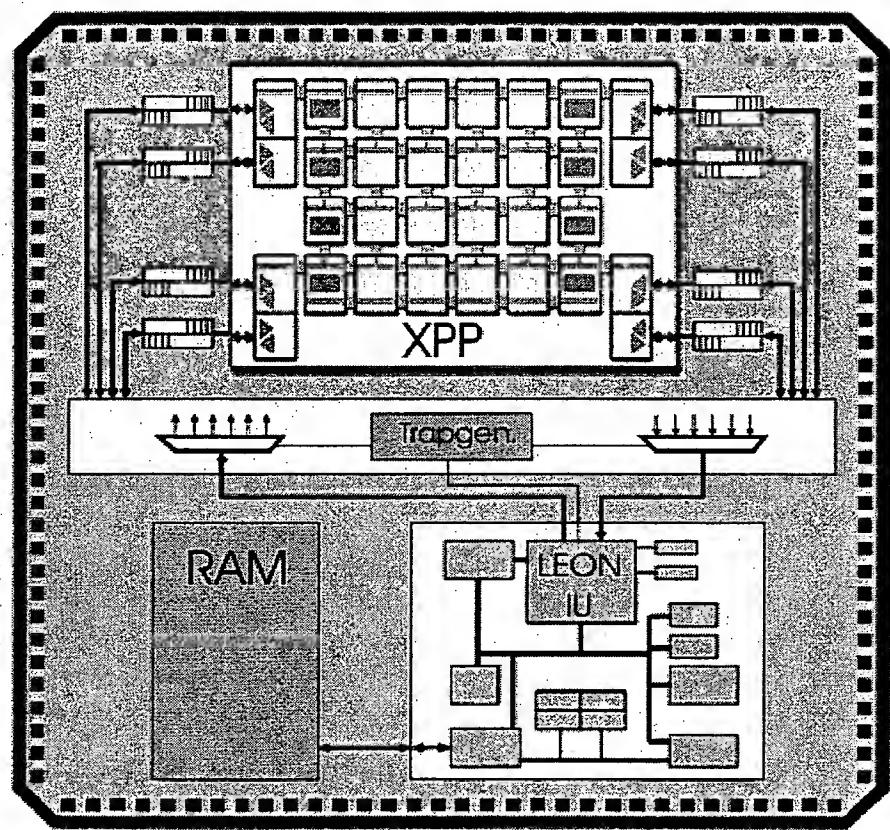


Fig. 52: Extended Datapath Overview

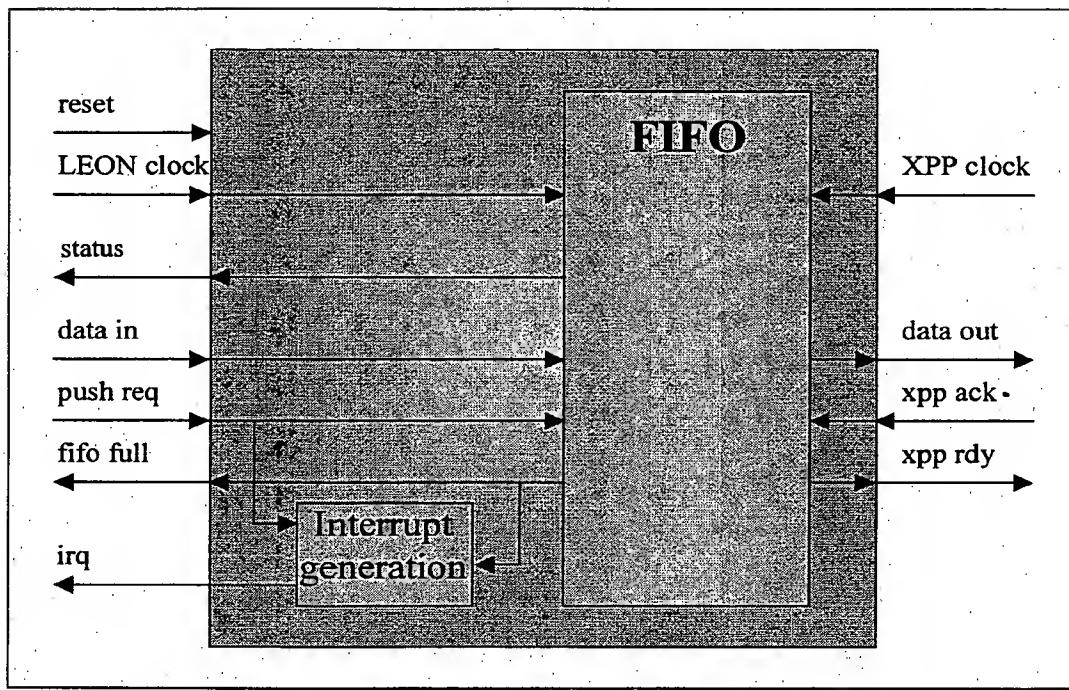
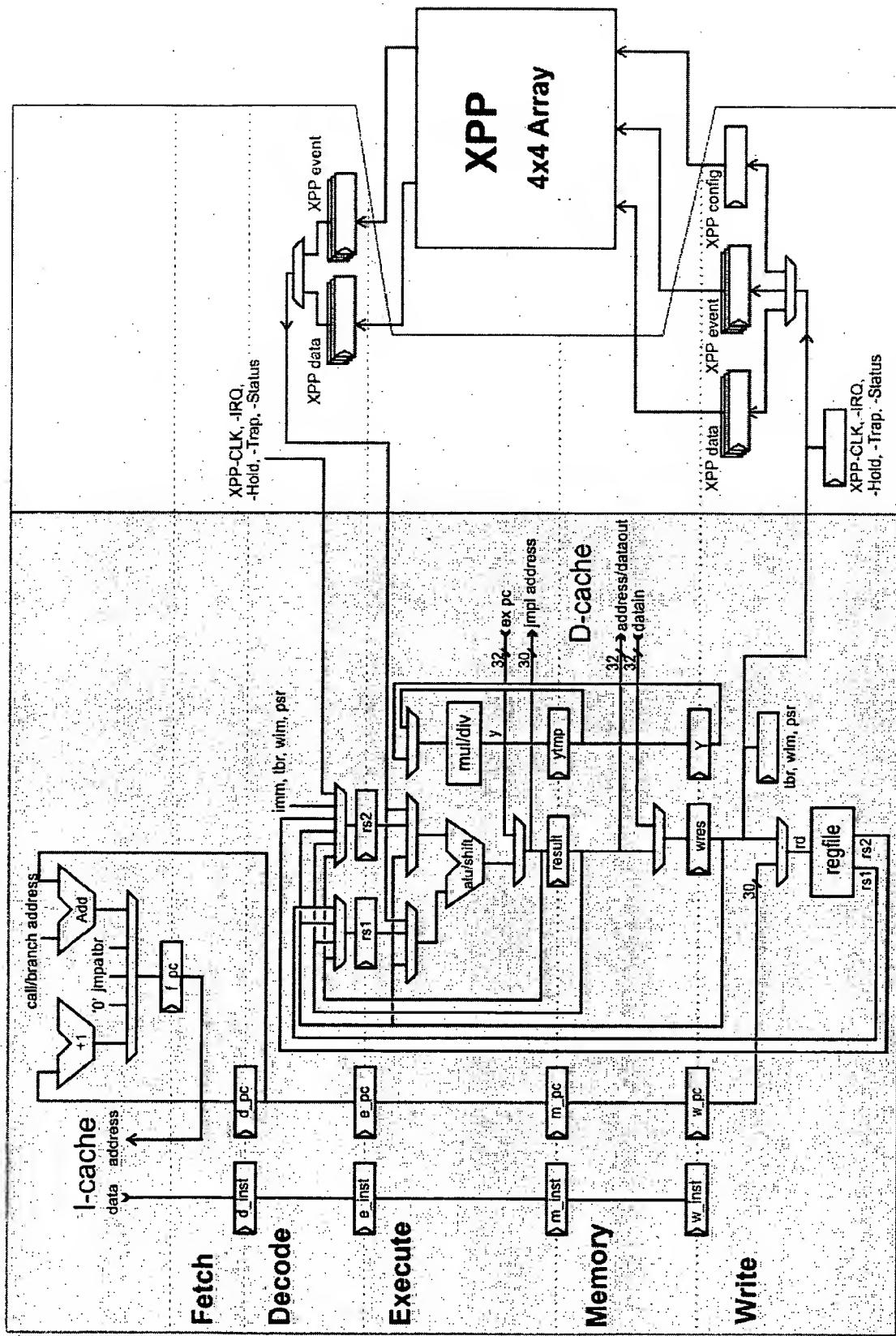


Fig. 53: LEON-to-XPP dual-clock FIFO



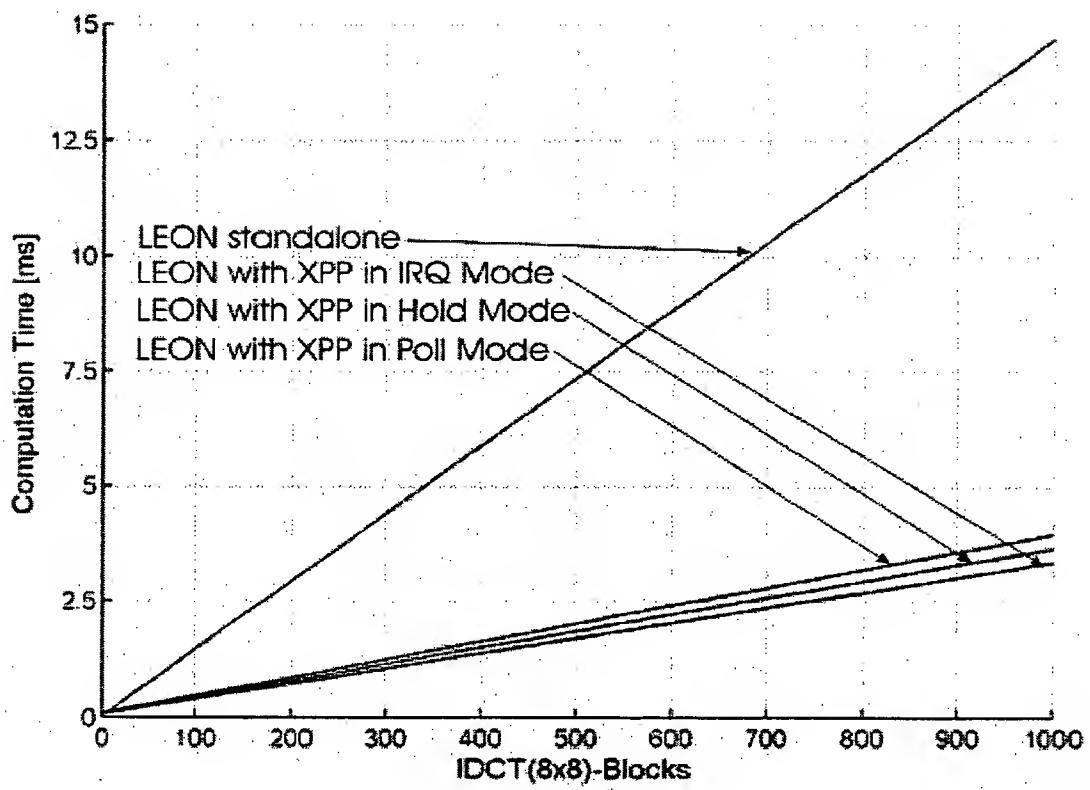


Fig. 55: Computation time of IDCT (8x8)

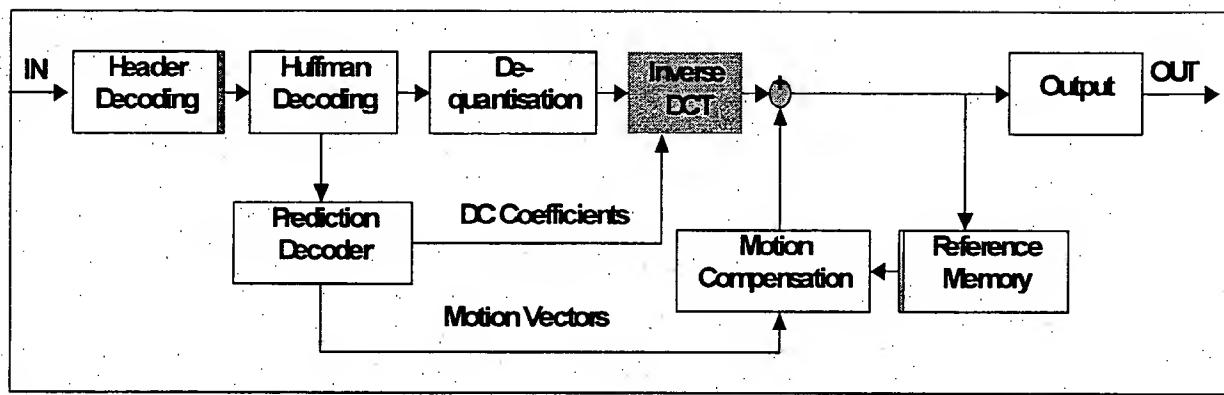


Fig. 56: MPEG-4 Decoder Blockdiagram

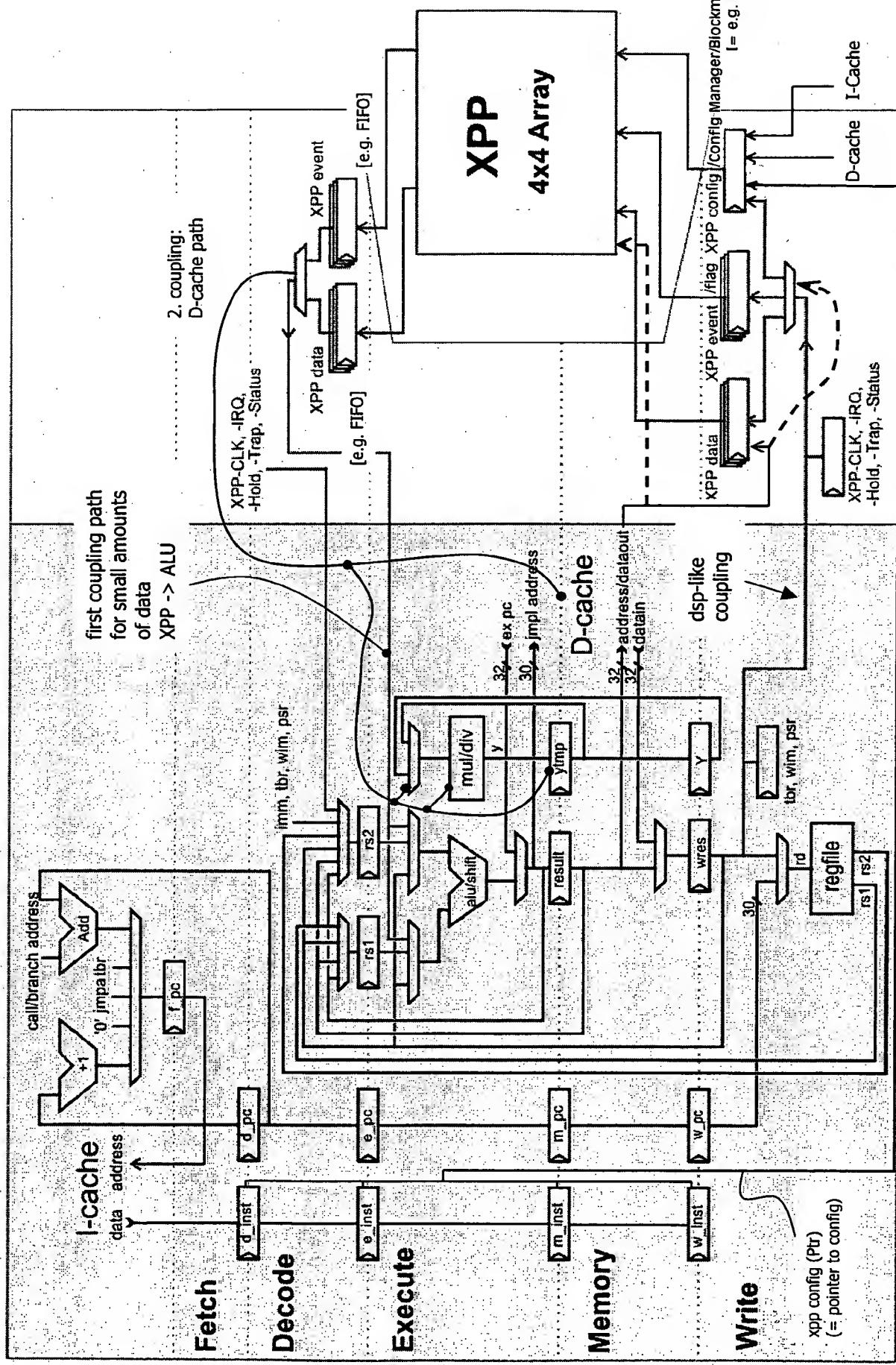


Fig. 57: Extended LEON Instruction Pipeline II

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.